REPORT DOCUMENTATION PAGE

FORM APPROVED
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	June 1998	

4. TITLE AND SUBTITLE OF REPORT
Causal Scheduling of Multiclass Traffic with Deadlines and Priorities

5. FUNDING NUMBERS
DAAH 04-95-1-0246

6. AUTHOR(S)
Pierre-Francois Seri

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Coordinated Science Laboratory
University of Illinois
1308 W. Main Street
Urbana, IL 61801

8. PERFORMING ORGANIZATION REPORT NUMBER
UILU-ENG-98-2211

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
U.S. Army Research Office
P. O. Box 12211
Research Triangle Park, NC 27709-2211

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
33988-EL-FRI

11. SUPPLEMENTARY NOTES:
The views, opinions and/or findings contained in this report are those of the author and should not be construed as an official Dept. of the Army position, policy or decision, unless so designated by other documentation.

12a. DISTRIBUTION AVAILABILITY STATEMENT
Approved for public release; distribution unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

This thesis analyzes causal scheduling and scheduling-dropping policies in a discrete time model. Packets from different priority classes arrive with arbitrary deadlines. The packets must be scheduled before their deadlines. We present three sets of results. The first set is divided into two parts. We first characterize all causal scheduling policies which maximize the throughput whatever the sequence of packets arriving to be scheduled. We then extend the analysis to causal scheduling-dropping policies, that is, scheduling policies which can drop packets before their expiration without scheduling them. We characterize all such scheduling policies that maximize the throughput and those that do so while minimizing the buffer occupancy. The second set of results considers a two-class model. We characterize all causal scheduling and scheduling-dropping policies that maximize the high-priority class throughput subject to maximizing the overall throughput. For the scheduling-dropping policies, we characterize those policies that minimize the buffer occupancy. We finally analyze the more general multiclass case, characterize all causal scheduling policies that optimize two novel optimality criteria, and present two simple scheduling-dropping policies that optimize those criteria.

14. SUBJECT TERMS
Scheduling, deadlines, priorities

15. NUMBER OF PAGES: 56

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT: UNCLASSIFIED	18. SECURITY CLASSIFICATION OF THIS PAGE: UNCLASSIFIED	19. SECURITY CLASSIFICATION OF ABSTRACT: UNCLASSIFIED	20. LIMITATION OF ABSTRACT: UL

CAUSAL SCHEDULING OF MULTICLASS TRAFFIC WITH DEADLINES AND PRIORITIES

BY

PIERRE-FRANCOIS SERI

ABSTRACT

This thesis analyzes causal scheduling and scheduling-dropping policies in a discrete time model. Packets from different priority classes arrive with arbitrary deadlines. The packets must be scheduled before their deadlines. We present three sets of results. The first set is divided into two parts. We first characterize all causal scheduling policies which maximize the throughput whatever the sequence of packets arriving to be scheduled. We then extend the analysis to causal scheduling-dropping policies, that is, scheduling policies which can drop packets before their expiration without scheduling them. We characterize all such scheduling policies that maximize the throughput and those that do so while minimizing the buffer occupancy. The second set of results considers a two-class model. We characterize all causal scheduling and scheduling-dropping policies that maximize the high-priority class throughput subject to maximizing the overall throughput. For the scheduling-dropping policies, we characterize those policies that minimize the buffer occupancy. We finally analyze the more general multiclass case, characterize all causal scheduling policies that optimize two novel optimality criteria, and present two simple scheduling-dropping policies that optimize those criteria.

ABSTRACT

This thesis analyzes causal scheduling and scheduling-dropping policies in a discrete time model. Packets from different priority classes arrive with arbitrary deadlines. The packets must be scheduled before their deadlines. We present three sets of results. The first set is divided into two parts. We first characterize all causal scheduling policies which maximize the throughput whatever the sequence of packets arriving to be scheduled. We then extend the analysis to causal scheduling-dropping policies, that is, scheduling policies which can drop packets before their expiration without scheduling them. We characterize all such scheduling policies that maximize the throughput and those that do so while minimizing the buffer occupancy. The second set of results considers a two-class model. We characterize all causal scheduling and scheduling-dropping policies that maximize the high-priority class throughput subject to maximizing the overall throughput. For the scheduling-dropping policies, we characterize those policies that minimize the buffer occupancy. We finally analyze the more general multiclass case, characterize all causal scheduling policies that optimize two novel optimality criteria, and present two simple scheduling-dropping policies that optimize those criteria.

ACKNOWLEDGMENTS

I would like to thank my advisor, Professor Bruce Hajek, for his patient and very kind guidance during my work on this thesis.

I also thank Professor Sawarte, who, with Professor Hajek, gave me the opportunity to be part of this program.

I would also like to thank my wife for her support while I was working on this thesis.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Multimedia applications are increasingly popular for communication media which were originally designed to transmit homogeneous traffic such as voice traffic. The heterogeneous nature of multimedia traffic makes it necessary to develop new transmission strategies, strategies that take into account new traffic properties such as real-time constraints and different priority levels. This thesis focuses on scheduling multiclass traffic with priorities and hard deadlines. It builds on the work already presented in [1]. The model analyzed has been used extensively in the literature. It provides a very simple—yet realistic—representation of a system in which a single server is used to multiplex traffic from several input streams into a single output stream (as is the case in a node of an ATM switch), and it provides the opportunity to tackle some of the basic and fundamental questions related to discrete real-time scheduling with priorities.

The model is time-slotted and consists of a server, which schedules packets from diverse input streams, and a buffer in which packets are stored prior to scheduling. Each source transmits packets with deadlines, which are assumed to be *arbitrary*, by which the packets must be scheduled. A packet that is not scheduled before its deadline expires and is simply dropped. Packets are of equal length and have equal service time, which is the the unit of time. It is further assumed that only one such packet can be served at a time and that in each time slot the new arrivals appear shortly before the scheduling decision. Packets are buffered prior to their scheduling or expiration.

Our model is clearly related to the discrete-time uniprocessor scheduling of tasks with priorities and deadlines, which has been extensively studied in the computer science literature (see [2]). This thesis, however, does not make any assumptions on the nature

of the incoming traffic. It can be *arbitrary*, whereas most papers in the computer science literature assume either a complete a priori knowledge of the incoming traffic or a set of conditions on its shape and distribution. This thesis ignores such hypotheses and focuses on the design of efficient *causal* (on-line or real-time) scheduling policies—policies where the decision is based *solely* on the past and the present. This thesis also considers scheduling policies that can drop packets before their expiration without scheduling them. They are referred to as *scheduling-dropping* policies. In fact, all scheduling policies could be seen as scheduling-dropping policies, but the distinction is made here because early dropping of packets raises particular analytical issues.

The contributions of our work are manifold. They can be divided into three sets of results. In the first set, priorities are ignored. We characterize all causal scheduling and scheduling-dropping policies (note that for a causal scheduling-dropping policy, both the scheduling and dropping decisions are made on-line) that maximize, over *all* scheduling policies, the number of packets served in any interval of time starting in time slot 1, and do so whatever the incoming traffic. Those policies are called *throughput optimal* (TO) scheduling or scheduling-dropping policies. Among such scheduling-dropping policies, we characterize those that minimize the buffer occupancy: the minimization is done time slot by time slot (sample-pathwise). Several studies of have analyzed conditions, necessary or sufficient, that, when satisfied by the incoming traffic, ensure that all packets can be scheduled by a particular scheduling policy without any loss [3], [4]. It is obvious that, in our model, any traffic that can be scheduled without loss by a given scheduling policy can also be scheduled without loss by any TO scheduling policy. Therefore, those policies maximize the server utilization and have the largest schedulability region for the traffic. That property justifies the emphasis placed on those policies in the remainder of this thesis. Also, scheduling-dropping policies are interesting for several because they reduce the buffer occupancy and allow the implementation of an early negative acknowledgment procedure for the packets dropped. Such a procedure has been shown to improve the overall throughput in an end-to-end transmission of video data [5]. Neither the study of end-to-end transmission nor the effect of early negative acknowledgments on the overall

traffic are in the scope of our work, but the results of [5] and [6] shows the importance of scheduling-dropping policies.

We would like to stress that to our knowledge there has never been either a complete and *accurate* analysis of a TO optimal scheduling dropping policy or an attempt to analytically study the effect of early dropping on the throughput achieved by those policies. Furthermore, very few simple TO scheduling policies have been described in the literature. The S-OPT algorithm described in [7] and [8] is a TO optimal scheduling policy, and the earliest deadline first (EDF) scheduling policy has been shown to be throughput optimal [9]. Most studies or practical implementations related to real-time scheduling with deadlines generally use the EDF scheduling policy, because it is both throughput optimal and very simple. However, that policy is not efficient in a setting where both throughput optimality and prioritizing are important: it does not consider priorities—which can be essential—in its scheduling decisions. It is therefore important to design alternative TO scheduling policies that take priorities into account to select the packets to be scheduled.

The importance of the characterization of all causal TO scheduling and scheduling-dropping policies is clearly reflected in the second set of results presented in this paper. Those results are obtained for a two-class model: packets arriving either belong to class 1 (high priority) or to class 2 (low priority). The classes are disjoint. The goal of our work in this setting was to determine how much prioritizing can be achieved without losing throughput optimality. It is clearly not possible to causally optimize both prioritizing and throughput optimality. The following simple example illustrates that point. Let us assume that in a time slot there are two packets in the buffer; one packet has *laxity* 1 and low priority, and the other laxity 2 and high priority, where the laxity of a packet is the amount of time left before its expiration. Then, to be throughput optimal, a causal scheduling policy must schedule the lower priority packet, otherwise the packet would expire and the policy would serve at most one packet if there were no subsequent arrivals. On the other hand, to optimize prioritizing, the policy must schedule the higher priority packet, otherwise it would lose at least one high-priority packet if such a packet

were to arrive with laxity one in the next time slot. Of course, the subsequent arrivals are unknown to any causal scheduling policy. Therefore, the future of or additional information on the characteristics of the traffic are necessary to optimize both criteria simultaneously; they are not available in our model.

The tradeoff between throughput optimality and prioritizing is illustrated on a larger scale by Figure 1.1. It shows the throughput region for a two-class model. The simulation covers 100 000 time slots. Packets of class 1 and class 2 arrive according to Poisson processes with intensity 0.5. The laxity of each packet upon arrival is uniformly distributed over 1, 2, 3, and the number of packets of each class scheduled by five different policies is pictured. Point SP1 = (48163, 30310) in the figure corresponds to the throughput pair achieved by a causal "static priority" scheduling policy that maximizes the throughput of class 1 packets and, subject to that maximization, maximizes the throughput of class 2 packets. Point SP2 is similar, with the priorities swapped. Point EDF=(41131, 40824) is the throughput pair achieved by the EDF scheduling policy, which is throughput optimal. Point MOSTO1 = (45429, 36526) is the throughput pair achieved by a scheduling policy that maximizes the throughput of class 1 packets, subject to being a causal TO scheduling policy: it is referred to as a MOSTO scheduling policy. Point MOSTO2 is equivalent to point MOSTO1, when the priorities are swapped. Note that the line segments linking two points in the figure are on the boundary of the achievable throughput region.

We characterize all MOSTO scheduling and scheduling-dropping policies, and among them, the scheduling-dropping policies which minimize the buffer occupancy. The existence of a *causal* MOSTO scheduling policy has never been shown before, and it is not as obvious as that of a causal TO scheduling policy. We refer the reader to the formal proof in [1], where the difficulties in showing that existence are clearly stated. Also, it is noteworthy that our results show that the minimum buffer occupancy required, slot by slot, for a causal MOSTO and TO scheduling policies are identical.

The final set of results presents natural extensions of previous results to a general multiclass case: we consider a model with more than two classes. When there are more

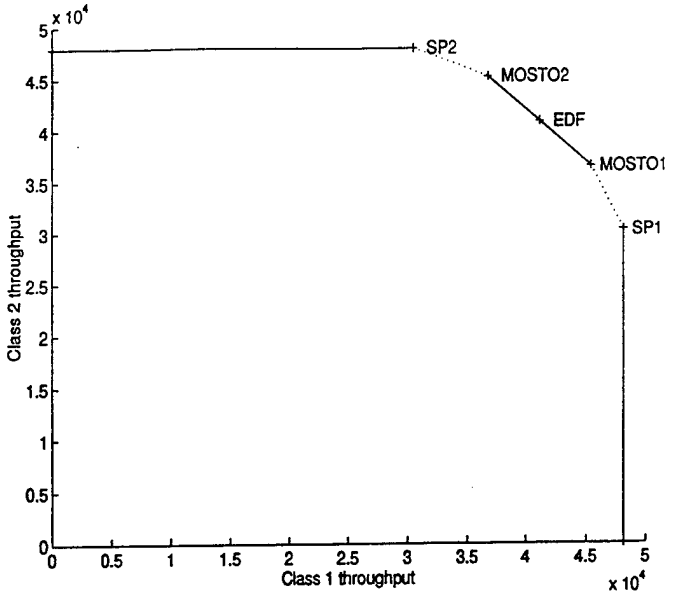

Figure 1.1 Throughput region for the two class model.

than two classes, several optimality criteria can be considered. We focus on two novel and important criteria: *nested throughput optimality* (NTO) and *static priority throughput optimality* (SPTO). The statement of those criteria is presented in Chapter 2. Like the MOSTO criterion, they are hierarchical criteria, that is, criteria that optimize a quantity subject to another optimality condition. Causal NTO and SPTO scheduling-dropping policies are described. They minimize the buffer occupancy over all causal TO scheduling-dropping policies, and their complexity, in a time slot, is $O(M)$ per new arrival, where M is the number of packets already in the buffer. That complexity is similar to the complexity obtained for the simplest implementation of the EDF scheduling policy.

Chapter 2 presents the contributions described earlier and gives the notations used later in the proofs; Chapter 6 synthesizes and concludes our discussion; and the chapters in between present the complete proofs of all formal results given in this thesis.

CHAPTER 2

STATEMENT OF RESULTS

2.1 Terminology

As already stated, packets with deadlines and priority classes *may* arrive in any time slot, prior to the scheduling decision. It takes one unit of time to serve a packet. The deadline $d(p)$ of a packet is the last time slot in which it can be scheduled. The class $C(p)$ of a packet p is a positive integer. In the monoclass case, all packets are class 1 packets. It is also assumed that the lower the class number, the higher the priority. An *arrival sequence* $\mathcal{A}$ is a sequence $A = (A_n : n \geq 1)$ such that A_n is the set of packets arriving in slot n (it could be an empty set). The sets in the sequence are mutually disjoint, and it is assumed that $d(p) \geq n$ for any packet $p \in A_n$. A *schedule* for an arrival sequence is a sequence $(p_n : n \geq 1)$ such that for each positive n, either $p_n = \Delta$ (which indicates that no packet is scheduled in time slot n), or $p_n \in A_1 \cup \ldots \cup A_n$ and $n \leq d(p_n)$. We also assume that packets, after being scheduled, leave the system never to return. Thus, no packet appears more than once in a schedule. A *scheduling policy* π gives a schedule $(\pi(\mathcal{A}; n) : n \geq 1)$ for each arrival sequence $\mathcal{A}$. The scheduling policy is *causal* if $\pi(\mathcal{A}; n)$ depends only on $A_1, \ldots, A_n$. We also define the following sets in order to describe the evolution of the system: given an arrival sequence $\mathcal{A}$ and a schedule $(p_n : n \geq 1)$,

- $R_n; n \geq 0$, is the set of packets remaining in the system at the end of slot n. The set does not include packets that have already been scheduled, or that have expired. We assume that $R_0 = \emptyset$.

- $S_n; n \geq 1$, is the set of all packets that are in the system in time slot n, prior to dropping and scheduling.

- $E_n; n \geq 1$, is the set of packets that expire at the end of slot n without being dropped or scheduled.

- $Q_n; n \geq 1$, is the subset of packets, among packets in S_n, retained by a scheduling-dropping policy in time slot n, prior to scheduling. It includes the packet scheduled in time slot n, if any, and is equal to S_n for a nondropping scheduling policy.

Let us consider p_n to be the packet scheduled in slot n, possibly $\triangle$. The following evolution equations then hold: for $n \geq 1$,

$$S_n = R_{n-1} \cup A_n \tag{2.1}$$

$$Q_n \subset S_n \tag{2.2}$$

$$E_n = \{p \in Q_n - p_n : d(p) = n\} \tag{2.3}$$

$$R_n = Q_n - E_n - p_n \tag{2.4}$$

All the sets defined above, except those in the arrival sequence, depend on the scheduling policy π used to schedule the packets; that dependence will be stressed, when necessary, by rewriting Q_n as $Q_n(\pi)$, for example. As already stated in Chapter 1, the *laxity* of a packet is the amount of time left before its expiration: in time slot k, the laxity of a packet p is $l(p) = d(p) - k + 1$. All packets in the system have positive laxities. Since, in any time slot, the laxities in a set of packets are the true measure of the scheduling possibilities for that set, we will sometimes consider a set of packets with laxities without explicit mention of the current time slot; it is implicitly the present.

2.2 Throughput Optimal Scheduling Policies

A scheduling policy π is said to be *throughput optimal* (TO) if for any arrival sequence $\mathcal{A}$, for any $n \geq 1$, π schedules at least as many packets in slots $1, \ldots, n$ as any other scheduling policy.

	Example 1	Example 2	Example 3
S	1 2 5 5 5	3 3 6	1 2 2 4 4 4 6
Positions i	1 2 3 4 5	1 2 3	1 2 3 4 5 6 7
$\delta(i)$	0 0 2 1 0	2 1 3	0 0 -1 0 -1 -2 -1
i^*, l^*	$i^* = 1, l^* = 1$	$i^* = 2, l^* = 3$	$i^* = 6, l^* = 4$

Figure 2.1 Illustration or the computation of $\Phi_s(S)$.

Let S be a set of packets with laxities. The set $\Phi_s(S)$ (the subscript denotes the single class case) is then defined as follows: we first write S as the ordered set $\{p_1, \ldots, p_r\}$ so that $l(p_1) \leq l(p_2) \leq \ldots \leq l(p_r)$, and

$$\delta(i) = l(p_i) - i \quad \text{for } 1 \leq i \leq r \tag{2.5}$$

$$\delta_{min} = min\{\delta(i) : 1 \leq i \leq r\} \tag{2.6}$$

$$i^* = min\{i : \delta(i) = \delta_{min}\} \tag{2.7}$$

$$l^* = l(p_{i^*}) \tag{2.8}$$

$$\Phi_s(S) = \{p_1, \ldots, p_{i^*}\} \tag{2.9}$$

The definition above is applied to three examples in Figure 2.1. It is clear that the set $\Phi_s(S)$ defined earlier has cardinality i^*, that it is the set of all packets in S with laxity smaller than or equal to l^*, and that it does not depend on how packets with the same laxity are ordered: it is thus a well-defined subset of S.

Theorem 2.1 *A causal scheduling policy π is TO if and only if for any arrival sequence $\mathcal{A}$ and any positive* k *such that $S_k \neq \emptyset$, $\pi(\mathcal{A}; k) \in \Phi_s(S_k(\pi))$.*

We call $\Phi_s(S_k)$ the *no regret scheduling set* of S_k, because, whatever the future arrivals, a packet $p \in S_k$ can be scheduled in time slot k without loss of future throughput if and only if $p \in \Phi_s(S_k)$. Specifying such a decision set is equivalent to characterizing all causal TO scheduling policies.

2.3 Properties of a Set of Packets

The properties of a set of packets described in this section are fundamental to understanding the formalism developed in the remainder of this thesis. They reflect the matroidal properties of a set of packets with laxities (see [1], [10] and [11] for more details). Let S be a set of packets with laxities and let τ be a finite subset of the positive integers. Then, τ is an index of a set of time slots, relative to the "current" time slot k, and $t \in \tau$ corresponds to time slot $t + k - 1$. We say that S *can cover* τ if there exists an assignment of a subset of packets (possibly all) in S to elements in τ such that (a) all elements in τ are assigned a packet in S, (b) the packets assigned to those elements are distinct, and (c) the laxity of the packet assigned to a particular element t in τ is greater than or equal to t (i.e., if there exists a schedule of packets in S which schedules a packet in each time slot included in τ). A set of packets S is said to be *schedulable* if it can cover a set τ such that $|S| = |\tau|$ (i.e., they have equal cardinality). Given a set of packets S, $rank(S)$ is defined as the largest cardinality of a schedulable subset of S. A subset of S is a *maximum cardinality schedulable* subset of S if it is schedulable and has cardinality rank(S). Let A and B be two sets of packets with laxities. We write $A \succeq B$, and say that A *can cover* B if A can cover any set τ that B can cover. We write $A \equiv B$ if $A \succeq B$ and $B \succeq A$, and say that A is *equivalent* to B.

Lemma 2.1 *Among all the sets that a given set of packets S can cover, there exists the "largest and latest" one, which we denote by $\mathcal{L}(S)$, such that for any set τ that S can cover, $|\tau| \leq |\mathcal{L}(S)|$, and the ith largest element in τ is smaller than or equal to the ith largest element in $\mathcal{L}(S)$, for $1 \leq i \leq |\tau|$. $\mathcal{L}(S)$ is, in a sense, the* latest optimal *schedule of S: it shows how much delay can be introduced in the scheduling of successive packets in S while serving the maximum number of those packets.*

Proof. Let $V = \{q_1, \ldots, q_{\text{rank}(s)}\}$ be a subset of packets in S with the rank(S) largest laxities. Clearly, for any j such that $1 \leq j \leq \text{rank}(S)$, the j elements with largest laxities in V can cover any set that can be covered by a subset of j packets in S. Let us define the following set $\tau = \{t_1, \ldots, t_{\text{rank}(S)}\}$, where for any i with $1 \leq i \leq \text{rank}(S)$, t_i

is the largest integer so that there exists a set of $\text{rank}(S) - i + 1$ packets in V that can cover $\{t_i, t_i + 1, \ldots, t_i + \text{rank}(S) - i\}$. Then $\tau = \mathcal{L}(S)$.

□

Lemma 2.2 *Let S be a set of packets with laxities and let Q be a subset of S. Then $Q \equiv S$ if and only if $rank(S) = rank(Q)$. In particular, Q minimizes $|Q|$ subject to $[Q \subset S$ and $Q \equiv S]$ if and only if Q is a maximum cardinality schedulable subset of S.*

Proof. The "only if" part of the first assertion in the lemma is simple. Let us now turn to the "if" part: we assume that $S \succeq Q$, and $\text{rank}(S) = \text{rank}(Q) = r$, and those two sets are not equivalent. Let us write $\mathcal{L}(S) = \{s_1, \ldots, s_r\}$, and $\mathcal{L}(Q) = \{t_1, \ldots, t_r\}$ (the sets are written with their elements in increasing order). From our assumption, we know that $\mathcal{L}(S)$ strictly dominates $\mathcal{L}(Q)$ in the sense that there exists j such that $1 \leq j \leq r$ and $t_j < s_j$. Let j^* be the largest such j, and i^* be the smallest integer i in $[1, r]$ such that $t_{j^*} < s_i$. Let us also consider a schedule $\mathcal{P} = \{p_1, \ldots p_r\}$ (respectively $\mathcal{P}' = \{q_1, \ldots, q_r\}$) of elements in S (respectively in Q) which covers $\mathcal{L}(S)$ (respectively $\mathcal{L}(Q)$); then the set of packets $\{q_1, \ldots, q_{j^*}\}$ and $\{p_{i^*}, \ldots, p_r\}$ are disjoint, and their union is a schedulable set that can be scheduled in the time slots indexed by $\{t_1, \ldots, t_{j^*}, s_{i^*}, \ldots, s_r\}$. Their union is also a set with cardinality strictly larger than $\text{rank}(S)$, which is a contradiction since $Q \subset S$.

The second assertion of the theorem is implied by the first.

□

The proof above implies the matroidal property of schedulable sets: it basically shows either that two sets of packets with the same rank are equivalent or that their union is a set of packets with a rank strictly larger than their individual ranks. We refer the reader to [1] and [10] for more details on the subject. Two simple algorithms for the construction of maximum schedulable subsets and latest optimal schedules are also described in [1].

2.4 Throughput Optimal Scheduling-Dropping Policies

2.4.1 Characterization of the policies

The results in this section are related to policies that drop some packets before their expiration without serving them. Such a policy π is causal if and only if for any arrival sequence $\mathcal{A} = (A_1, \ldots, A_n, \ldots)$ for any positive n, $\pi(\mathcal{A}; n)$ and $Q_n(\pi)$ depend only on $A_1, \ldots, A_n$. As was the case for nondropping scheduling policies, a scheduling-dropping policy is TO if, for any positive n, it maximizes the number of packets scheduled in time slots $1, \ldots, n$, over all scheduling policies. We next characterize *all* causal TO scheduling-dropping policies.

Theorem 2.2 *A causal scheduling-dropping policy π is TO if and only if for any arrival sequence $\mathcal{A}$ and all positive k, $Q_k \equiv S_k$, and $\pi(\mathcal{A}; k) \in \Phi_s(Q_k)$ if $S_k \neq \emptyset$. Furthermore, if π and π' are both causal TO scheduling policies, $Q_k(\pi) \equiv Q_k(\pi')$ for all k.*

Theorem 2.2 allows us to identify those causal TO scheduling-dropping policies that minimize, slot per slot, the number of packets carried over from one slot to the next. Indeed, from the second assertion of the theorem, $\text{rank}(Q_k(\pi))$ does not depend on the causal TO scheduling policy π considered: it is only a function of $\mathcal{A}$ and k, and thus can be rewritten as $r^*(\mathcal{A}; k)$ for all such policies. Therefore, for all those policies $|Q_k| \geq r^*(\mathcal{A}, k)$, and by selecting Q_k to be a maximum cardinality schedulable subset of S_k, one minimizes the buffer occupancy in slot k. That last result is formally stated in the following corollary to Theorem 2.2:

Corollary 2.1 *For any arrival sequence $\mathcal{A}$ and any $k \geq 1$, a causal TO scheduling-dropping policy π minimizes $|Q_k|$ over all such policies if and only if Q_k is schedulable.*

Figure 2.2 shows the average queue size (time average of Q_k over 10 000 time slots) for various mean arrival rates, assuming that laxities on arrival are uniformly distributed on $\{1, \ldots, 9\}$. The upper curve is generated by the EDF policy (without dropping) and

the lower curve is generated by the Dropping EDF policy, which is described in the next section. One can see that for large values of the mean arrival rates, the upper curve grows roughly linearly with slope 5, while the lower curve asymptotically approaches 10. It means that if the node is congested, the buffer occupancy of the standard EDF will grow linearly while that of the Dropping EDF essentially stays the same. We note that a *genie* scheduling-dropping policy which has prior knowledge of the future arrivals can be TO while reducing the buffer occupancy below the level achieved by the best causal TO scheduling-dropping policies.

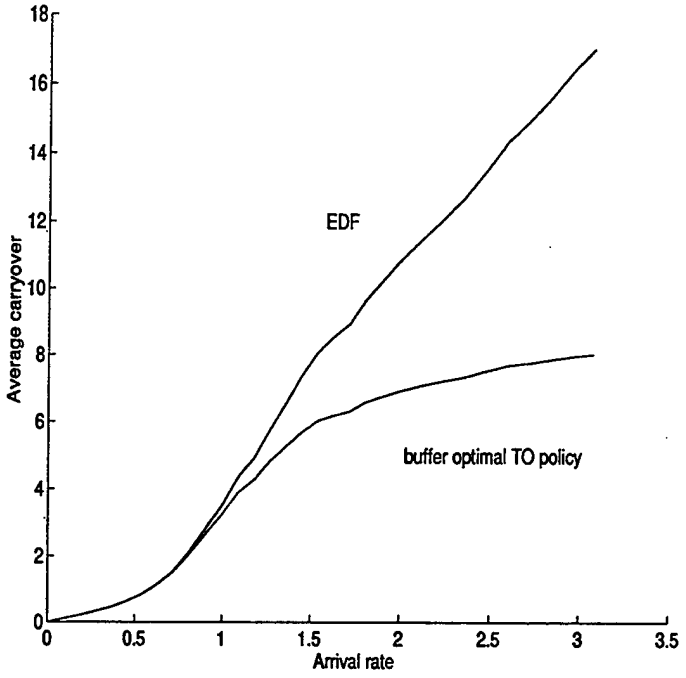

Figure 2.2 Average buffer occupancy.

2.4.2 The Dropping EDF scheduling policy

This section describes a dropping version of the EDF policy. We will assume that there exists a total order on the packets such that if two packets have the same laxity, it is still possible to sort them according to a fixed order. In this section, we will write that $p < q$ and say that packet p (respectively q) is smaller (respectively larger) than

packet q (respectively p), if $l(p) > l(q)$ or $[l(p) = l(q)$, and p precedes q in the total order]. Note that this assumption does not affect the definition of the EDF scheduling policy. A causal scheduling or scheduling-dropping policy is completely defined if for any arrival sequence $\mathcal{A}$ and any positive time slot n, its dropping and scheduling decisions are specified as functions of the set of packets carried over from the previous time slot (R_{n-1}; empty in the first time slot) and the new arrivals (A_n).

For the Dropping EDF scheduling policy, it is assumed that the packets carried over from the previous slots are in the buffer, sorted in increasing order, according to the order specified earlier. The Dropping EDF scheduling policy consists of two procedures: the EDF-DROP dropping procedure (which specifies the packets that are retained) and the scheduling procedure (which simply schedules the packet at the head of the buffer and then moves the remaining packets by one position toward the head of the buffer). The dropping procedure is completely described in Figures 2.3. The buffer is specified as an array of packets; the packet in position i is referred to as $BUFFER[i]$. Also, it is assumed that $BUFFER[1]$ is the packet at the head of the buffer, if any; the tail is towards the higher indices. Briefly, the dropping procedure attempts to insert new packets in the buffer sequentially, one packet at a time. When a new packet can be added to the set of buffered packets without violating the buffer's schedulability, the packet is inserted in the buffer so that the increasing order is preserved. Otherwise, the procedure drops the largest (in the sense defined earlier) packet already buffered and the new one, which—if excluded—would make the remaining set of packets schedulable. The remaining packets are inserted in the buffer so that the increasing order is preserved. The scheduling procedure is then the usual EDF scheduling policy, applied to the set of packets remaining in the buffer after the insertion of all new arrivals.

The Dropping EDF policy has a complexity equivalent to that of the standard EDF. However, in the case of a highly congested node, the Dropping EDF policy, is more efficient since it drops packets that would occupy buffer space and add to the ordering complexity. In the description above, we mention the fact that the Dropping EDF policy drops the largest packet that can be dropped without affecting future optimality. The

EDF-DROP Dropping Procedure

Input: $A_n = \{q_1, \ldots, q_m\}$ (arbitrary order) : the set of the new arrivals.

```
EDF-Drop(A_n)
begin

        i = 1;
        while(i ≤ |A_n|)

                Insert(q_i);
                i = i + 1;

        end while

end
```

Insert subprocedure

```
Insert(packet)
begin

        Temp = packet; k = 1;
        while(1)

                if (l(Temp) < k)

                        drop Temp; and exit;

                else if (BUFFER[k] = empty)

                        BUFFER[position] = Temp; and exit;

                else if (BUFFER[k] > Temp)

                        p = BUFFER[k]; BUFFER[k] = Temp; Temp = p; and k = k + 1;


                else

                        k = k + 1;

        end while

end
```

Figure 2.3 The dropping procedure for the Dropping EDF policy.

choice of the largest packet does not bear any particular meaning in the algorithm. It yields a simpler description of the algorithm. Also, the Dropping EDF policy is equivalent to algorithm G1 presented in [1].

Claim 2.1 *Dropping EDF is a TO scheduling-dropping policy.*

Proof. The dropping procedure is a greedy algorithm which, in each time slot $n \geq 1$, starts with a schedulable subset R_{n-1} of S_n and adds packets from A_n as long as the resulting set is schedulable. Otherwise, the algorithm drops a packet so that the rank is preserved. Thus, the algorithm retains in each time slot $n \geq 1$ a maximum cardinality schedulable subset Q_n of S_n (see [1] and [10] for a proof). Also, the scheduling procedure schedules a smallest laxity packet in that subset, which, of course, is a packet in $\Phi_s(S_n)$. Then, Theorem 2.2 and Lemma 2.2 imply the result.

□

2.5 MOSTO Scheduling Policies

We now consider the two-class model. Packets arriving belong to one of two classes: class 1 or class 2. Class 1 packets are of higher priority. A causal scheduling policy π is said to be *maximum class one subject to throughput optimality* (MOSTO) if it is TO and if for any arrival sequence $\mathcal{A}$ (note that $\mathcal{A}$ is now a sequence of arrivals of packets from both classes) and any positive n, the scheduling policy π schedules a least as many class 1 packets in time slots $1, \ldots, n$ as would any other causal TO policy. We put the emphasis on the following points: first, the optimization here is made over all causal TO scheduling policies since we already showed in Chapter 1 that the optimization cannot be generalized to all scheduling policies; also, it is not clear that a causal MOSTO scheduling policy exists. We need to show that the following cannot happen: there exists some positive n and m with $n \neq m$ and a TO policy π such that, π maximizes the number of packets of class 1 served in time slots $1, \ldots, m$ over all causal TO scheduling policy, but does not maximize it in time slots $1, \ldots, n$, and any TO policy that serves as many

class one packets as π in time slots $1,\ldots,n$ does not maximize the number of class one packets served in the interval of time $1,\ldots,n$. That issue is of great importance since it constitutes the main hurdle in the characterization of MOSTO scheduling policies. The failure to consider similar questions has led several authors to give inaccurate proofs when analyzing scheduling policies in models similar to the one considered here. It is necessary to show that the decision made by a policy is optimal at the time of decision, *and* that it does not affect optimality in the future. An inductive approach is implicitly required.

Given a set of packets S, we use the following convention: $\overline{S}$ is the set of class 1 packets in S, and $\underline{S}$ is the set of class 2 packets in S. Also when representing a set of packets with laxities, the laxities are underlined or overlined according to the same rule. Let $\Phi_m(S)$ be the subset of S defined as follows:

$$\Phi_m(S) = \begin{cases} \Phi_s(S) \cap \Phi_s(\overline{S}) & \text{if } \overline{\Phi_s(S)} \neq \emptyset \\ \Phi_s(S) = \Phi_s(\underline{S}) & \text{if } \overline{\Phi_s(S)} = \emptyset \text{ and } S \neq \emptyset, \\ \emptyset & \text{if } S = \emptyset. \end{cases} \tag{2.10}$$

The set $\Phi_m(S)$ is the two-class equivalent of $\Phi_s(S)$, which was defined earlier, and can be seen as a MOSTO no regret scheduling set. The following theorem settles the issue of the characterization of all causal MOSTO scheduling policies:

Theorem 2.3 *A causal scheduling policy π is MOSTO if and only if for any arrival sequence $\mathcal{A}$ and positive k such that $S_k \neq \emptyset$, $\pi(\mathcal{A};k) \in \Phi_m(S_k(\pi))$.*

2.6 MOSTO Scheduling-Dropping Policies

2.6.1 Characterization of the policies

Given a set of packets S, define $\mathcal{K}(S)$ by $\mathcal{K}(S) = \mathcal{L}(C)$, where C is the set of packets in $\underline{S}$ with the $rank(S) - rank(\overline{S})$ largest laxities. Given two sets of packets S and T, write $S \asymp T$ if $[S \equiv T,\ \overline{S} \equiv \overline{T}$ and $\mathcal{K}(S) = \mathcal{K}(T)]$: we say that S and T are *strongly*

equivalent. The following theorem shows that two strongly equivalent sets are essentially "identical" when the MOSTO property is considered.

Theorem 2.4 *A causal scheduling-dropping policy π is MOSTO if and only if for any arrival sequence $\mathcal{A}$ and any positive k, $Q_k \asymp S_k$, and if $Q_k \neq \emptyset$ then $\pi(\mathcal{A};k) \in \Phi_m(Q_k)$. Furthermore, if π and π' are both causal MOSTO scheduling-dropping policies, then $Q_k(\pi) \asymp Q_k(\pi')$ for all positive k.*

Theorem 2.4 is the most important and significant contribution of our work. It extends the result from Theorem 2.2 to the two-class case: it clearly shows that the significant tradeoff between prioritizing and throughput optimality, which is achieved by a causal MOSTO scheduling policy does not require an overhead in the number of packets carried over from one slot to the next. The next corollary is the formal statement of the point made above.

Corollary 2.2 *For any arrival sequence $\mathcal{A}$ and any positive k, a causal MOSTO scheduling-dropping policy π minimizes $|Q_k|$ over all such policies if and only if Q_k is schedulable.*

The proof of the results above are presented in [1].

2.6.2 A causal MOSTO scheduling-dropping policy

In this section, we present the causal scheduling-dropping policy $D^{min-max}$, and give an intuitive explanation of the *virtual laxity*, which is presented below. Like the Dropping EDF scheduling-dropping policy, the $D^{min-max}$ scheduling-dropping, which is described in Figure 2.4, is divided into two procedures: a dropping procedure M-DROP, and a scheduling procedure, described in Figures 2.4 and 2.5. The data structure also consists of a buffer specified as an array of packets, but a new quantity, a *virtual laxity vector* (which is a length two vector) is assigned to each packet in the buffer. Thus, in this algorithm, each packet has a laxity and a class, which are policy independent, and a virtual laxity vector, which is assigned by the $D^{min-max}$ scheduling-dropping policy. The

```
D^{min-max}
Input: A_n = {q_1, ..., q_m} (arbitrary order) :  the set of the new arrivals.

begin

        M-DROP(A_n) :  {

                for (i = 1; i ≤ |A_n|; i++)

                        M-Insert(q_i);

                end for

        }
        Scheduling():  {

                if (BUFFER not empty)

                        transmit BUFFER[1];
                        move the remaining packets, if any, by one position towards BUFFER[1];
                        decrease the coordinates of the virtual laxity of each remaining packet by one;

        }

end
```

Figure 2.4 The algorithm $D^{min-max}$.

```
M-Insert(packet)
begin

        (L_packet, K_packet) = (l(packet), l(packet));
        Temp = packet;
        Temp_position = min(l(packet), |BUFFER|);
        FREE_ROLL: {
        while(L_Temp < L_BUFFER[Temp_position])

                Temp_position = Temp_position − 1;

        end while }
        while(Temp_position ≥ 1)

                if (L_Temp > L_BUFFER[Temp_position])

                        INSERTION_1: {
                        packets starting from position Temp_position + 1 are moved by one position towards the tail
                        of BUFFER; Temp is inserted at position Temp_position; exit; }

                else if (L_Temp = L_BUFFER[Temp_position])

                        EQUALITY: { if Temp and BUFFER[Temp_position] have different classes, then
                        insert the higher priority one at position Temp_position of BUFFER; the other one
                        becomes the new Temp; decrease L_Temp by one;
                        else if neither is of the lowest class and if they have different class virtual laxities,
                        then insert the packet with the larger class virtual laxity at position Temp_position of
                        BUFFER; the other one becomes the new Temp; then decrease L_Temp and Temp_position
                        by one;
                        else choose a packet arbitrarily between the packets and insert it at position
                        Temp_position in BUFFER, the other one becomes the new Temp, and decrease L_Temp
                        and K_Temp by one; }

        end while
        if (L_Temp > 0)

                INSERTION_2:{ call INSERTION_1;}

        else

                DROPPING: { move the packets in BUFFER by one position towards the
                tail, starting from the head of BUFFER, until a packet g is reached with L_g = l(g); drop g; increase
                by one the L coordinates of the virtual laxity vectors of Temp and the packets that have been moved,
                and, among these packets, increase by one the K coordinates of the virtual laxity vectors of the packets
                which are in the same class as g; insert Temp at position 1 of BUFFER; exit;}

end
```

Figure 2.5 The M-Insert subprocedure.

dropping procedure is applied in each time slot to the packets carried over from the previous time slots. Those packets are placed in the positions of the buffer according to a lexicographical order on the triplet $(L_p, C(p), K_p)$, where L_p and K_p are, respectively, the first and second coordinates of the virtual laxity vector of a packet p in the buffer. We call the first coordinate of the virtual laxity vector of a packet the *group virtual laxity* and the second coordinate the *class virtual laxity.* The variable $C(p)$ is the class of the packet.

The dropping procedure attempts to insert the new arrivals into the buffer, one new packet at a time. It either adds a new packet to the buffer without dropping any packet or drops either the new packet or a packet already buffered. At the same time, this procedure maintains the lexicographical order on the packets retained in the buffer. When all new packets, if any, have been processed and the buffer is not empty, the scheduling procedure is invoked. This procedure schedules the packet at the head of the buffer (note that the head of the buffer is still $BUFFER[1]$), moves the remaining packets by one position towards the head of the buffer, and decreases each coordinate of their virtual laxity vectors by one.

Before describing the intuition underlying the use of a virtual laxity, we will briefly discuss the complexity of the scheduling policies described so far. The $D^{min-max}$ scheduling-dropping policy has a complexity which is on average three times greater than that of the Dropping EDF scheduling-dropping policy. In each time slot and for each new arrival, assuming that there are already M packets in the buffer, the dropping procedure requires at most $3M$ operations, while the Dropping EDF procedure requires at most M operations. Also, if laxities on arrivals are assumed to be less than or equal to some positive B, as in [7], the two scheduling procedures have complexity $O(NB)$, where N is the number of new arrivals. Those complexities are much lower than the complexity $O(NB^2)$ per arrival presented in [7]; also, the complexity of the $D^{min-max}$ scheduling policy does not change when it is extended to the general multiclass case. Such an extension is presented later in this chapter.

Claim 2.2 *$D^{min-max}$ is a causal MOSTO scheduling policy.*

The proof of Claim 2.2 is deferred to Chapter 4. The construction of the dropping policy is based on the following remark: when two packets in a set of packets have the same laxity, the schedulability properties of that set are not affected if one of those packets has its laxity decreased by one. Decreasing the laxity by one yields a set of packets equivalent to the original one. Case A in Figure 2.6 illustrates that point. The process can then be applied repeatedly until all packets in the resulting set have distinct laxities. Case B in Figure 2.6 shows the result of applying the technique repeatedly to a particular set: the original and the final sets are still equivalent since they have the same latest optimal schedule. Since the laxities of packets do not depend on the scheduling policy, it is natural to assign them virtual laxities, which reflect the process of successive decreases of their laxities. However, in a multiclass model, if two packets of different classes have the same laxity, reducing one packet's laxity may affect the MOSTO property.

In case C, the class 1 packet has to be scheduled first in choice 1, while the class 2 packet has to be scheduled first in choice 2. It seems natural to reduce the virtual laxity of the higher priority packet in order to make it a more urgent packet. However, as shown in case D, the solution above cannot be applied systematically, since a high-priority packet might "virtually" expire in the process, even if all packets of that class can be scheduled. Those issues are settled in the M-Insert procedure: the high-priority packets have their virtual laxities decreased when there are equalities, but a special case is considered when such a packet reaches "virtual" expiration. Furthermore, the use of a single virtual laxity enables us to control the order in which packets can be served without losing the throughput optimality property. However, the MOSTO criterion requires two nested levels of optimality. The second coordinate of the virtual laxity vector enables us to apply the same laxity-reduction technique to packets in the same class, thus, it ensures the "optimality" of the scheduling of class one packets.

Original set		Equivalent set
$S = \{3, 4, 5, 5\}$		$S' = \{3, 4, 4, 5\}$
$\mathcal{L}(S) = \{2, 3, 4, 5\}$		$\mathcal{L}(S') = \{2, 3, 4, 5\}$

Case A: A set and its equivalent after one step.

Original set		Equivalent set
$S = \{3, 4, 5, 5\}$		$S' = \{2, 3, 4, 5\}$
$\mathcal{L}(S) = \{2, 3, 4, 5\}$		$\mathcal{L}(S') = \{2, 3, 4, 5\}$

Case B: A set and its final equivalent.

Original set		Equivalent set (choice 1)		Equivalent set (choice 2)
$S = \{\underline{3}, \overline{4}, \overline{5}, \overline{5}\}$		$S' = \{\overline{2}, \underline{3}, \overline{4}, \overline{5}\}$		$S'' = \{\underline{2}, \overline{3}, \overline{4}, \overline{5}\}$
$\mathcal{L}(S) = \{2, 3, 4, 5\}$		$\mathcal{L}(S') = \{2, 3, 4, 5\}$		$\mathcal{L}(S'') = \{2, 3, 4, 5\}$

Case C: Two differents equivalents of a set S.

Original set		Equivalent set
$S = \{\underline{2}, \underline{2}, \overline{2}\}$		$S' = \{\overline{0}, \underline{1}, \underline{2}\}$
$\mathcal{L}(S) = \{1, 2\}$		$\mathcal{L}(S') = \{1, 2\}$

Case D: An example of "virtual" expiration.

Figure 2.6 Intuitive illustration of virtual laxities.

2.7 Extension to the General Multiclass Case

In this section, we examine a model in which there are N classes of packets, class $1, \ldots, N$, where $N \geq 2$. We assume that the smaller the class, the higher the priority. We use the following convention: given a set of packets S, for any m such that $1 \leq m \leq N$, $S^{(m)}$ is the set of packets of class m in S. If $N \geq 3$, several optimality criteria can be defined. We propose two criteria, which, when applied to a two-class model, are simply the MOSTO optimality criterion studied earlier. For each criterion, we characterize all causal scheduling policies that satisfy the criterion, and we present a scheduling-dropping policy that satisfies the criterion. However, we do not characterize *all* scheduling-dropping policies which satisfy those criteria.

2.7.1 Static priority TO scheduling policies

A causal scheduling policy π is *order 1 static priority throughput optimal* (1-SPTO) if it is TO, and for any arrival sequence $\mathcal{A}$ (of N classes of packets, with $N \geq 2$) and any k, it schedules as many packets of class 1 as any causal TO scheduling policy in time slots $\{1, \ldots, k\}$—that is, if π is MOSTO when class 2 up to class N are grouped in a *super* low priority class.

For any n so that $2 \leq n \leq N-1$, a causal scheduling policy π is *order n* SPTO (n-SPTO) if it is $(n-1)$-SPTO, and for any positive k, schedules at least as many packets of class n as any causal $(n-1)$-SPTO scheduling policy in time slots $\{1, \ldots, k\}$.

A causal $(N-1)$-SPTO scheduling policy is referred to as a causal SPTO scheduling policy. The SPTO criterion is similar to a static priority criterion, where the throughput of the classes are successively maximized starting from the highest priority class; the only difference here is that the successive optimization is done subject to throughput optimality.

Given a set of packets S, let us define the set $\Phi_{SPTO}(S)$ as follows:

$$c^* = \min\{i : 1 \leq i \leq N, \text{ and } \Phi_s(S) \cap \Phi_s(S^{(i)}) \neq \emptyset \}, \quad \text{if } S \neq \emptyset \quad (2.11)$$

$$\Phi_{SPTO}(S) = \begin{cases} \Phi_s(S^{(c^*)}) \cap \Phi_s(S) & \text{if } S \neq \emptyset \\ \emptyset & \text{otherwise} \end{cases} \tag{2.12}$$

Note that the set defined above is equivalent to $\Phi_m(S)$, when $N = 2$. The following theorem characterizes all causal SPTO scheduling policies:

Theorem 2.5 *A causal scheduling policy π is an SPTO scheduling policy if and only if for any arrival sequence $\mathcal{A}$ and any k such that $S_k \neq \emptyset$, $\pi(\mathcal{A}; k) \in \Phi_{SPTO}(S_k)$.*

The set $\Phi_{SPTO}(S_k)$ is called the *SPTO no regret scheduling subset* of S_k. As was done for the TO and MOSTO optimality criteria, the definition of an SPTO scheduling policy can be extended to encompass scheduling-dropping policies. The characterization of all causal SPTO scheduling-dropping policies is not presented in this thesis. It could be the subject of an extension of this work. Nevertheless, we present a particular SPTO scheduling-dropping policy.

Claim 2.3 *$D^{min-max}$, when applied—as described earlier in this section—to the general multiclass case is an SPTO scheduling policy.*

2.7.2 Nested throughput optimal scheduling policies

The model is similar to the one considered above. There are N classes, where $N \geq 2$. The convention used to denote subsets of a particular class is also kept. We make the following definitions:

A causal scheduling policy π is *order 1 nested throughput optimal* (1-NTO) if it is TO, and for any arrival sequence and any k, it schedules at least as many packets of class in $\{1, \ldots, N-1\}$ as would any other TO scheduling policy in time slots $\{1, \ldots, k\}$.

For any n such that $2 \leq n \leq N-1$, a causal scheduling policy π is *order n* NTO (n-NTO) if it is $(n-1)$-NTO, and for any arrival sequence $\mathcal{A}$ and any positive k, it schedules at least as many packets of class in $\{1, \ldots, N-n\}$ packets as would any $(n-1)$-NTO scheduling policy in time slots $\{1, \ldots, k\}$.

A causal $(N-1)$-NTO scheduling policy is referred to as a causal NTO scheduling policy.

All the definitions above can of course be extended to encompass scheduling-dropping policies.

Given a set of packets S, let us define the set $\Phi_{NTO}(S)$ as follows:

$$U_i(S) = \bigcup_{j=1}^{i} S^{(j)} \quad \text{for } 1 \leq i \leq N \tag{2.13}$$

$$c^* = min\{i : 1 \leq i \leq N, \text{ and } \bigcap_{j=i}^{N} \Phi_s(U_j(S)) \neq \emptyset \} \quad \text{if } S \neq \emptyset \tag{2.14}$$

$$\Phi_{NTO}(S) = \begin{cases} \bigcap_{j=c^*}^{N} \Phi_s(U_j(S)) & \text{if } S \neq \emptyset \\ \emptyset & \text{otherwise} \end{cases} \tag{2.15}$$

Note that for $N = 2$, $\Phi_{NTO}(S) = \Phi_m(S)$. The next theorem characterizes all causal NTO scheduling policies:

Theorem 2.6 *A causal scheduling policy π is an NTO scheduling policy if and only if for any arrival sequence $\mathcal{A}$ and positive k such that $S_k(\pi) \neq \emptyset$, $\pi(\mathcal{A}; k) \in \Phi_{NTO}(S_k(\pi))$.*

The set $\Phi_{NTO}(S)$ is called the *NTO no regret scheduling subset* of S. We do not characterize all causal NTO scheduling-dropping policies, but we present a particular NTO scheduling-dropping policy, D^{nested}. The D^{nested} scheduling-dropping policy is similar to the $D^{min-max}$ scheduling-dropping policy, except that the data structure carried over from one slot to the next is larger; packets are now assigned a vector of length N, which contains N *virtual laxities*. The intuition underlying the use of virtual laxities is the same as in the $D^{min-max}$ scheduling-dropping policy: when two packets have the same laxity, we attempt to decrease the laxity of the highest priority by one. However, for the nested throughput optimality criterion, it is necessary to keep track of which classes of packets caused the decrease and by how much they decreased the laxity. This is achieved through the vector of virtual laxities. The scheduling policy is divided into two procedures: the NTO-DROP dropping procedure (which is described in Figures 2.7 and 2.8), and the scheduling procedure (which is performed after NTO-DROP). The scheduling procedure schedules the packet at the head of the buffer, moves the remaining packets by one position towards the head of the buffer, and decrease by one each coordinate of

```
D^nested
Input:  A_n = {q_1, ..., q_m} (arbitrary order) :  the set of the new arrivals.

begin

        NTO-DROP(A_n) :  {

                for (i = 1; i ≤ |A_n|; i++)
                        NTO-Insert(q_i);
                end for

        }
        Scheduling():  {

                if (BUFFER not empty)

                        transmit BUFFER[1];
                        move the remaining packets, if any, by one position towards BUFFER[1];
                        decrease the finite coordinates of the virtual laxity vector of the remaining packets by one;

        }

end
```

Figure 2.7 The algorithm D^{nested}

their virtual laxity vectors. Also, in the description of the procedures, we write L_p^i for the L^{th} coordinate of the virtual laxity of packet p.

Claim 2.4 *The scheduling policy D^{nested} is a causal NTO scheduling-dropping policy.*

NTO-Insert subprocedure

NTO-Insert($packet$)
begin

$Temp\_position = \min(l(packet), |BUFFER|)$;
$Temp = packet$;
Set the first $N - C(packet) + 1$ coordinates of the virtual laxity vector of $packet$ to $l(packet)$, and the remaining ones, if any, to ∞;
FREE_ROLL: {
while($L^1_{Temp} < L^1_{BUFFER[Temp\_position]}$)

$Temp\_position = Temp\_position - 1$;

end while }
while($Temp\_position \geq 1$)

if ($L^1_{Temp} > L^1_{BUFFER[Temp\_position]}$)

INSERTION_1: {
packets starting from position $Temp\_position + 1$ are moved by one position towards the tail of $BUFFER$; $Temp$ is inserted a position $Temp\_position$; exit; }

else if ($L^1_{Temp} = L^1_{BUFFER[Temp\_position]}$)

EQUALITY: { Compare the the packets $Temp$ and $BUFFER[Temp\_position]$ using the lexicographical order on the coordinates of their respective virtual laxity vectors ; the smaller one is inserted at $Temp\_position$ of $BUFFER$; the other one is the new $Temp$ (in case of equality, an arbitrary choice can be made); Let $n = max\{j : 1 \leq j \leq N$, and the first j coordinates of the virtual laxity vectors of both packets are identical$\}$; then decrease the first n coordinates of the virtual laxity vector of $Temp$ by one; $Temp\_position = Temp\_position - 1$; }

end while
if ($L_{Temp} > 0$)

INSERTION_2: call INSERTION_1;

else

DROPPING: { move the packets in $BUFFER$ by one position towards the tail, starting from the head of $BUFFER$, until a packet g is reached with $L^1_g = l(g)$; drop g; increase by one the $N - C(g) + 1$ first coordinates of the virtual laxity vectors of $Temp$ and the packets that have been moved; insert $Temp$ at position 1 of $BUFFER$; exit;}

end

Figure 2.8 The NTO-Insert subprocedure.

CHAPTER 3

ODDS AND ENDS

The proofs of Theorems 2.1, 2.2, 2.3, and 2.4 are presented in [1]. The results in this chapter are corollaries of those theorems and their proofs. Their presentation has been deferred to this chapter because they were not essential for understanding the previous material; they are mere implications of the analysis already given and are fundamental for the remaining proofs.

Lemma 3.1 *If* $A \subset S$ *and* $A \cap \Phi_s(S) \neq \emptyset$, *then* $\Phi_s(A) \subset \Phi_s(S)$.

See [1] for a proof.

Lemma 3.2 *If* $A \subset S$, *and* $\Phi_s(S) \subset A$ *then* $\Phi_s(S) = \Phi_s(A)$.

For any $B \subset S$, it is clear from the definition of Φ_s given in Section 2.4 that the following holds.

$$\Phi_s(S) \;=\; \Phi_s(S - \{p \in B : l(p) > l^*\}) \tag{3.1}$$

Where l^* is the largest laxity in $\Phi_s(S)$. This is true since the minimization will again be achieved at laxity l^*. The packets with laxity strictly greater than l^* that are not excluded in the right hand side of Equation (3.1) either have their rank decreased or preserved when the packets are ordered in increasing order of laxity. The packets with laxity smaller than or equal to l^* keep their rank. The result follows.

Let us assume that there exists a sequence of time slots $\mathcal{T} = (t_n : n \geq 1)$ (possibly finite) such that for any $n \geq 1$, the server goes on vacation in time slot t_n: it means that no scheduling can be completed in time slot t_n. $\mathcal{T}$ can be arbitrary and may be a

priori unknown to causal scheduling policies. We now extend the definition of MOSTO scheduling policies to a model with vacations. We say that a scheduling policy is MOSTO in a model with vacations if it satisfies the conditions of Theorem 2.1, and, subject to that, maximizes the number of class 1 packets served in any interval of time starting in the first time slot (note that the TO optimality ignores the vacations). The following lemma is then implied by the proofs of Theorems 2.1 and theorem 2.3:

Lemma 3.3 *In a model with vacations, a causal scheduling policy π is MOSTO if (sufficient condition) for any arrival sequence $\mathcal{A}$, any sequence $\mathcal{T}$ of vacation times, and any positive k such that $S_k \neq \emptyset$, $\pi(\mathcal{A}; k) \in \Phi_m(S_k)$.*

The proofs of Theorems 2.1 and 2.3 are based on the covering properties of the sets of packets in the systems; the equivalence properties are cleary preserved even if vacation times occur. Therefore, the induction arguments developed in the proofs of the "if" part of both theorems are still valid: those induction hypotheses rely only on covering properties of the set of packets in the system. The "only if" part of the theorems are no longer valid since their proofs rely on the construction of particular sequences of future arrivals. It can easily be seen that if all the future time slots are vacation time slots, a scheduling policy which has made no-regret scheduling decisions until the current time slot only needs to serve a packet of the highest priority among the packets it has in its system to be TO (or MOSTO). The implications of Lemma 3.3 are significant for the characterization of causal SPTO and NTO scheduling policies.

CHAPTER 4

PROOFS OF CHARACTERIZATION THEOREMS FOR N CLASSES

In this section we prove Theorems 2.5 and 2.6.

4.1 Proof of Theorem 2.5

Proof. The proof is by induction. The induction hypothesis is the following one:

$$\mathcal{P}(N): \text{ the theorem is true for } N \text{ classes}$$

The proposition $\mathcal{P}(2)$ is true from Theorem 2.3. Let us assume that $\mathcal{P}(N)$ is true for some $N \geq 2$. Let us now consider a model with $N+1$ classes.

We begin with the "if" part of the theorem. Let π be a scheduling policy which satisfies the characterization of the theorem. We need to show that π is an N-SPTO scheduling policy. Recall that a causal SPTO scheduling policy π (for the $N+1$ class case; it is referred to as an N-SPTO scheduling policy to stress the fact that it schedules $N+1$ classes of packets), if such a policy exists, is by definition a causal $(N-1)$-SPTO scheduling policy for the following N class model: class N and class $N+1$ packets are grouped in a *super*-low-priority class, which is referred to as class N^*, while the other classes are preserved. Thus, π is a causal $(N-1)$-SPTO scheduling policy for the N class model. We simply need to show that it maximizes the throughput for class-N packets. For any arrival sequence $\mathcal{A}$ of packets (from the $N+1$ classes), the induction hypothesis, which states the validity of the theorem for an N-class model, specifies the time slots in which an N-SPTO scheduling policy, if such a policy exists, can schedule packets

from the super-low-priority class: this is true since all $(N-1)$-SPTO scheduling policies schedule packets of the same class in each time slot. Let us consider the time slots in which these policies schedule packets from the $N-1$ highest priority classes as virtual vacation time slots for a model which consists of class N and $N+1$. For any time slot v_k in which the $(N-1)$-SPTO scheduling policies schedule a packet of class N^*, the packet scheduled is in $\Phi_s(S_k^{(N)} \cup S_k^{(N+1)})$; this is true from $\mathcal{P}(N)$ and Lemma 3.2. It also follows that Lemma 3.3 can be applied since the packets scheduled are in $\Phi_s(S_{v_k})$. We then know that a causal $(N-1)$-SPTO scheduling policy for the N class model that schedules a packet in $\Phi_m(S_k^{(N)} \cup S_k^{(N+1)})$ (which as shown above is included in $\Phi_s(S_k)$) in any time slot k that is not a virtual vacation time slot is an N-SPTO scheduling policy. Thus, π is a causal N-SPTO scheduling policy since, by definition, for any time slot k in which π schedules a packet of class N or $N+1$, π schedules a packet in $\Phi_m(S_k^{(N)} \cup S_k^{(N+1)})$. It should be noted that Φ_m is used here by implicitly considering that class N and class $N+1$ form a two class *sub*model. Thus, we have a sufficient condition for a causal scheduling policy π to be N-SPTO.

It remains to show that an $(N-1)$-SPTO policy π has to satisfy the necessary condition (the "only if" part) of the theorem to be a causal N-SPTO scheduling policy. Let us assume that a $(N-1)$-SPTO scheduling policy π makes decision identical to that of an N-SPTO scheduling policy in time slots prior to time slot k for some k. For the sake of argument by contradiction, let us assume that in time slot k policy π does not schedule a packet in $\Phi_{SPTO}(S_k)$. From the induction hypothesis, if it happens in a time slot where $\Phi_{SPTO}(S_k)$ contains a packet of one of the first $N-1$ high-priority classes, then π is not a causal N-SPTO scheduling policy since it is not a causal $(N-1)$-SPTO scheduling policy for the N class model with class N and $N+1$ grouped in a super-low-priority class. Thus, let us assume that k is such that $\Phi_{SPTO}(S_k) \subset (S_k^{(N)} \cup S_k^{(N+1)})$. We know that in that case $\Phi_{SPTO}(S_k) = \Phi_m(S_k^{(N)} \cup S_k^{(N+1)})$. Thus, two cases are possible:

Case 1. Policy π schedules a packet of class $N+1$ while $\Phi_m(S_k^{(N)} \cup S_k^{(N+1)})$ contains a packet of class N: then clearly π is not N-SPTO since it is beaten (in the current time slot) by a policy satisfying the sufficient condition of the theorem.

Case 2. The set $\Phi_m(S_k^{(N)} \cup S_k^{(N+1)})$ contains a class-N packet, and π schedules a class-N packet in $\Phi_s(S_k^{(N)} \cup S_k^{(N+1)})$ (as it must by $\mathcal{P}(N)$) which is not in $\Phi_m(S_k^{(N)} \cup S_k^{(N+1)})$. It means that π schedules a packet which is not in $\Phi_s(S^{(N)})$. Then, we use a subsequent arrival sequence similar to the one used in the proof of the "only if" part of Theorem 2.1. The arrival sequence is constructed as follows: in the next $l-1$ time slots, where l is the highest laxity of a packet in $\Phi_s(S_k^{(N)})$, a class-1 packet arrives with laxity 1. Those packets are clearly in $\Phi_{SPTO}(S_m(\delta))$, for $k+1 \leq m \leq k+l$, for any scheduling policy δ. Let π' be a scheduling policy that has decisions identical to π in the time slots preceding k, that schedules a packet in $\Phi_{SPTO}(S_k(\pi))$ in time slot k, that schedules the new arrivals in the next $l-1$ time slots, and that schedules a packet in $\Phi_{SPTO}(S_n(\pi'))$ in any time slot n with $n > l-1$. The policy π' is clearly an N-SPTO scheduling policy. We need to show that π' serves more packets of class smaller than or equal to N than π. We know that all class-N packets in $\Phi_s(S_k(\pi))$ with deadline larger than $k+l-1$ (i.e, the class-N packets in $\Phi_s(S_k(\pi))$ that are not in $\Phi_s(S_k(\pi)^{(N)})$) and the packets that are in $S_k(\pi) - \Phi_s(S_k(\pi))$, which includes all packets of class smaller than N, if any, can be scheduled after time slot $k+l-1$. We now prove the following claim:

Claim 4.1 *Let us assume that there are $N+1$ classes in the system: class $0, 1, \ldots, N$ (respectively class $1, \ldots, N+1$). Let $\mathcal{A}$ be an arrival sequence such that $A_1^{(0)}$ (respectively $U_N(A_1)$) is schedulable and $A_n = \emptyset$ for $n > 1$, then for any causal N-NTO (respectively N-SPTO) scheduling policy π, π schedules all packets of $A_1^{(0)}$ (respectively $U_N(A_1)$).*

We present the proof for an N-NTO scheduling policy. The proof for an N-SPTO scheduling policy is similar.

We say that a set S of packets with laxities is *critical* if S is schedulable but is not schedulable starting in the next time slot. We base the proof of Claim 4.1 on two facts.

Fact 1. For any set S, if $S^{(0)}$ is critical, then $\Phi_{NTO}(S) \subset S^{(0)}$. If a packet of class 0 is not scheduled in the current time slot, one such packet will be lost by the policy. The fact also follows from the following remark which is simple to verify. If a set of packets with laxities R is critical, then there exist l such that there are l packets in R with laxity

smaller than or equal to l. Clearly, if a set of packets with laxities R is critical, then for any other set of packets with laxities U, $\Phi_s(RUS) \cap R \neq \emptyset$ (see the characterization of Φ_s in Section 2.4).

Fact 2. For any schedulable set of packets with laxities R, R is critical in the time slot corresponding to the smallest value in $\mathcal{L}(R)$. This has to be true from the definition of $\mathcal{L}(R)$ given in Section 2.3.

Thus, the N-NTO scheduling policy π of Claim 4.1 schedules a packet of class 0 before the time slot corresponding to the smallest value in $\mathcal{L}(A_1^{(0)})$. Since, π schedules a packet in $\Phi_s(A_1^{(0)})$, the packets remaining after π schedules the first packet of class 0, if any, form a schedulable set in the time slot following that scheduling. Then, the reasoning can be extended inductively to show the validity of the claim.

Thus, from Claim 4.1, all packets in $U_N(S_k)$ with laxity greater than or equal to $k+l$ are scheduled by π'. In time slots after time slot k, policy π schedules at least one class-N packet less than π' because, in time slot k, π scheduled a class-N packet which could have been scheduled later.

□

4.2 Proof of Theorem 2.6

Proof. The proof of Theorem 2.6 is similar to the one given for Theorem 2.5. We also use an inductive approach. We begin with the "if" part of the theorem. The induction hypothesis is:

$$\mathcal{P}(N): \text{ the theorem is true for } N \text{ classes}$$

The proposition $\mathcal{P}(2)$ is true since the NTO criterion is the MOSTO criterion and Φ_{NTO} is Φ_m for a two-class model. Let us assume that the hypothesis is true for some N greater than or equal to 2. We show that it is still true for $N+1$. The $N+1$ classes are classes $0, 1, \ldots, N$. The technique is similar to the one adopted earlier: the $N+1$ class model is reduced to an N class model by regrouping class 0 and class 1 in a *super*-high-priority class 1^*. Then, we use the fact that, by definition, a causal N-NTO scheduling policy

for the $N+1$ class model (which will always be referred to as a N-NTO scheduling policy in this section), if such a policy exists, is an $(N-1)$-NTO scheduling policy for the model consisting of class $1^*, 2, \ldots, N$. All causal $(N-1)$-NTO scheduling policies schedule packets of the same class in each time slot. Let us consider the time slots in which these policies serve packets from the $N-1$ lowest priority classes as virtual vacation time slots for a model which consists of class 0 and class 1. Thus, a causal scheduling policy π is a causal N-NTO scheduling policy if it satisfies the conditions of Theorem 2.6 for $N-1$ classes, and in the time slots which are not virtual vacation time slots it serves as many packets of class 0 as would any other causal scheduling policy which satisfies the conditions of the theorem (i.e., any other $(N-1)$-NTO scheduling policy for the N-class model). Let π be an $(N-1)$-NTO policy. We know from the hypothesis that in a time slots k in which π schedules a packet from class 0 or class 1 (i.e., class 1^*), π schedules a packet in $\Phi_{NTO}(S_k) = \Phi_s(S_k^{(0)} \cup S_k^{(1)}) \cap (\bigcap_{i=2}^{N} \Phi_s(U_i(S_k)))$ since $U_{1^*}(S_k) = S_k^{(0)} \cup S_k^{(1)}$, and $c^* = 1^*$. From Lemma 3.3, we know that π is MOSTO for the two class model consisting of class 0 and class 1 with vacation times if it schedules packets in $\Phi_m(S_k^{(1)} \cup S_k^{(2)})$. But, by definition, in the time slots in which π schedules a packet of class 1^*, $\Phi_{NTO}(S_k)$ (for $N+1$ classes) $= \Phi_{NTO}$ (for N classes) $\cap\, \Phi_m(S_k^{(0)} \cup S_k^{(1)})$ is included in $\Phi_m(S_k^{(0)} \cup S_k^{(1)})$. Thus, a policy π which satisfies the condition of the hypothesis for $N+1$ classes is a causal N-NTO scheduling policy.

It remains to show that it is necessary for an N-NTO scheduling policy π to satisfy those conditions. Let us assume that π is a $(N-1)$-NTO scheduling policy that has made decisions identical to that of an N-NTO scheduling policy prior to some time slot k. For the sake of argument by contradiction, let us assume that π does not satisfy the conditions of Theorem 2.6. By the induction hypothesis, π has to satisfy the condition of $\mathcal{P}(N)$ for the modified N class model. Thus, there are only two possible cases:

Case 1. In a time slot k in which there is a class 0 packet in $\Phi_{NTO}(S_k)$, π schedules a class 1 packet in that set. Then π is not NTO since it is beaten by a policy which satisfies the sufficient condition above.

Case 2. In a time slot in which there is a class 0 packet in $\Phi_{NTO}(S_k)$, π serves a class 0 packet which is in $\Phi_{NTO}(S_k)$ for the N class model, but is not in $\Phi_s(S_k^{(0)}$. Then, we construct a subsequent arrival sequence like the one in the proof of the "only if" part of Theorem 2.5 above, and a causal scheduling policy π' similar to the one described in that proof, with the difference that the arrivals are now of class 0, and that π' now schedules packets in $\Phi_{NTO}(S_n(\pi'))$ in any time slot n after time slot $k+l-1$, where l is now the highest laxity of a packet in $\Phi_s(S_k^{(0)}$. The policy π' is a causal N-NTO scheduling policy. We now show that π' beats π by showing that π' serves more packets of class 0 than π in time slots after time slot $k-1$. We know that the packets of class 0 that are not in $\Phi_s(S_k^{(0)}$ (i.e, the packets of class 0 with laxity larger than l), if any, can be scheduled in time slots after time slot $k+l-1$. We simply need to prove that π' schedules those packets. From Claim 4.1, we know that policy π' described earlier schedules all packets of priority higher than or equal to N that have laxity greater than or equal to $k+l$. Policy π loses one such packet. □

A few remarks need to be made about the proofs given in the sections of this chapter: Lemma 3.3 is used in the proof even though the vacation times seen by the reduced model can depend on the scheduling decisions made by policies on that reduced model. This is possible since we assume that the general policy (i.e., the policy scheduling the $N+1$ classes of packets) is $(N-1)$-SPTO or $(N-1)$-NTO. In that case, the vacation times seen by the reduced model do not depend on the scheduling choice, since all $(N-1)$-SPTO or $(N-1)$-NTO scheduling policies serve packets of the same class in each time slot. Thus, the theorem can be applied. However, it would not be true in a general framework if the vacation times depend on the scheduling decisions.

CHAPTER 5

VERIFICATION OF THE ALGORITHMS

We prove Claims 2.2, 2.3, and 2.4.

5.1 Proof of Claim 2.2

In this section, we consider the two-class model. We adopt the following convention in order to simplify the notation: the packet at position k of $BUFFER$ is referred to as B_k; that is, B_k is equal to $BUFFER[k]$. Also, in the M-Insert procedure, the variable *C_position* is associated with a packet *C_packet*, which is the packet the procedure attempts to insert in $BUFFER$. We say that a packet p is *located* at position k of $BUFFER$, if $p = B_k$ or if $p =$ *C_packet* and *C_position* $= k$. Clearly, only *C_packet* can be located at position 0 of $BUFFER$. Of course, a packet p is said to be located after (respectively before) another packet q if the position where p is located is larger (respectively smaller) than the position where q is located. We further consider that the data structure of the algorithm consists of the triplet (*C_position*, *C_packet*, $BUFFER$), and we call packets in the data structures the packets which are in $BUFFER$ and *C_packet*, if any. We assume that *C_packet* is equal to Δ before or after a call to the M-Insert procedure. Then, the following properties hold before and after any call to the M-Insert and Scheduling subprocedures :

Property 1: For any packet q in $BUFFER$, $L_q \geq 1$.

Property 2: For any packet p in the data structure of the algorithm so that if *C_packet* $\neq \Delta$, p is either *C_packet* or a packet at a position greater than *C_position*;

if on the other hand $C\_packet = \triangle$, then p is any packet in $BUFFER$, the following holds: $\underline{V}(p) = (L_p, K_p)$, where

$$L_p = l(p) - \left(\begin{array}{l} \text{the number of packets } q \\ \text{located after } p \text{ in } BUFFER \text{ such that} \\ C(p) \leq C(q) \text{ and } L_q \leq l(p) \end{array} \right) \tag{5.1}$$

$$K_p = l(p) - \left(\begin{array}{l} \text{the number of packets } q \\ \text{located after } p \text{ in } BUFFER \text{ such that} \\ C(p) = C(q) \text{ and } K_q \leq l(p) \end{array} \right) \tag{5.2}$$

Property 3: For any packets q and p in the data structure such that q is located after p, $L_p < L_q$, and if p and q also have the same class, then $K_p < K_q$.

In the Property 2, we considered the case when $Temp \neq \triangle$ in order to simplify the proof for the M-Insert procedure. We now prove that the properties hold. To do so, we use an inductive approach. We assumed that the properties hold before a call to M-Insert or *Scheduling*, we want to show that they hold after that call. In the case of M-Insert, we use a second level of induction (i.e, we consider the validity of the properties during the execution of that procedure). We show that the properties hold as long as $C\_packet \neq \triangle$ in M-Insert, which simply yields the validity of the higher level induction. Thus, for M-Insert, we are mainly concerned with the lower level induction.

We prove Property 1. Clearly, if that property holds before the insertion of a new packet is attempted, it holds at any time during that attempt. Therefore, the M-Drop procedure does not affect the validity of the property, if one assumes that it was true before a call to that procedure was made. Also, from Property 3, which is proved below, it is clear that the scheduling procedure does not affect Property 1.

We now show that Properties 2 and 3 hold. It is clear that the scheduling procedure does not affect those properties, since they are based on the packets located after a packet in $BUFFER$, and the scheduling procedure removes the packet at the head of the $BUFFER$. Also, the decrease in the coordinates of the virtual laxity vectors of packets in the $BUFFER$ mirrors the increase in time after the scheduling, and thus

preserves the properties. We simply need to focus on the effects of the M-Drop procedure on the validity of the properties. We show that for any fixed time slot n, any packet q_i from the set of packets arriving in time slot n, and any value taken by the triplet $(C\_position, C\_packet, BUFFER)$ during the insertion of q_i in $BUFFER$, the properties hold. Since we already showed that the scheduling procedure and the increase in time do not affect the properties, we only need to prove the result just mentioned for a fixed time slot, and a trivial inductive argument shall settle the question. Thus, without loss of generality, we fix the "present" time slot, and work with packets with laxities. We note that the set of instructions labeled FREE_ROLL in the description of the procedure in Figure 2.5 do not affect the properties: they are just used to make sure that the packet q_i starts to contend at the right position of $BUFFER$. Thus, we will only consider the part of the procedure below those instructions, and will assume that any call to those instructions has already been completed. We also note that if the packet to be inserted q_i is such that there is no packet p in $BUFFER$ with $L_p = l(q_i)$, then the set of instructions labeled $INSERTION\_1$ is invoked, and the packet q_i is just inserted in $BUFFER$ with both coordinates of its virtual laxity vector equal to its laxity. The virtual laxity vectors of the other packets in $BUFFER$, if any, are not modified in the process. Also, the variable *C_packet* remains equal to q_i until the insertion of the packet in $BUFFER$. Thus, before insertion (this must be stressed: once it is inserted in $BUFFER$, it is no longer a contender), we say that *C_packet* is an *isolated contender*. The following results are simple implications of the Properties 1, 2 and 3:

Implication 1: For any packet q in the data structure (*C_packet* included) such that q is not an isolated contender, q is located after *C_position* if $C\_packet \neq \Delta$ and $q \neq C\_packet$, there is a packet $r(q)$ in $BUFFER$ such that: $q = r(q)$ or q is located before $r(q)$, $L_{r(q)} = l(q)$, the L coordinates of the virtual laxity vectors of the packets located after q and before $r(q)$ are consecutive, and those packets and $r(q)$ have priority lower than or equal to that of q.

We prove the implication: the implication is true for any packet that is inserted in $BUFFER$ just after having been an isolated contender, and clearly stays true for the

other packets in $BUFFER$, if any, after the insertion of that isolated contender. Thus, we consider the validity of the implication for nonisolated contenders. Property 3 tells us that $r(q)$ is uniquely defined if it exists. Let q' be the last packet (last starting from the head of $BUFFER$ in the location sense) such that $L_{q'} \leq l(q)$. From Property 3, we have

$$L_{q'} \geq L_q + |\{ \text{ packets located between } q \text{ and } q', q \text{ excluded } \}| \tag{5.3}$$

$$\geq L_q + \left| \left\{ \begin{array}{l} \text{packets } p \text{ located between } q \text{ and } q', q \text{ excluded,} \\ \text{with } C(p) \geq C(q) \end{array} \right\} \right| \tag{5.4}$$

$$\geq l(q) \tag{5.5}$$

where the last inequality is implied by Property 2. Thus, we see that

$$L_{q'} = l(q) \tag{5.6}$$

$$\left| \left\{ \begin{array}{l} \text{packets located between} \\ q \text{ and } q', q' \text{ included} \end{array} \right\} \right| = \left| \left\{ \begin{array}{l} \text{packets } p \text{ located between} \\ q \text{ and } q', q' \text{ included} \\ \text{with } C(p) \geq C(q) \end{array} \right\} \right| \tag{5.7}$$

Thus $q' = r(q)$, and the coordinates of the virtual laxities are consecutive for packets located between q and $r(q)$; they are actually consecutive on the set of packets consisting of q, $r(q)$, and the packets located between q and $r(q)$; equation (5.7) shows the result pertaining to the priorities.

Implication 2: For any packet q in the data structure (*C_packet* included) such that q is not an isolated contender, q is located after *C_position* if *C_packet* $\neq \Delta$ and $q \neq$ *C_packet*, there is a packet $\tilde{r}(q)$ in $BUFFER$ such that: $q = \tilde{r}(q)$ or q is located before $\tilde{r}(q)$, $K_{\tilde{r}(q)} = l(q)$, $\tilde{r}(q)$ is of the same class as q, and the class virtual laxities of the class one packets of class $C(q)$ located after $\tilde{q}$ and before $\tilde{r}(q)$ are consecutive.

We do not prove this implication since it is basically similar the Implication 1. Therefore, the steps of the proofs are the same except for the restriction of the analysis to packets of the same class.

We now return to the proof of the validity of the properties. We use an inductive approach. We assume that the properties are true at some stage in the M-Drop procedure, where a stage is characterized by a value of the triplet $(C\_position, C\_packet, BUFFER)$, and we then proceed to show that they are still true at any later stage of the procedure. We actually only need to show that the properties hold at the next stage w reached after a call to one of the sets of instructions labeled in the M-Insert procedure in Figure 2.5. There are five such sets of instructions. We therefore consider five cases. In the first case, we assume that the next stage is reached after a call to the set of instructions labeled FREE_ROLL. Then, we have already settled the question earlier, and the propertiers still hold.

In the second case, we assume that the next stage comes after a call to the set of instructions labeled INSERTION_1 (i.e., C_packet is inserted in $BUFFER$ just after its current location $C\_position$). Then two subcases are possible: if there is no packet p in $BUFFER$ with $L_p = l(C\_packet)$. That is, if C_packet is an isolated contender, then from what was done earlier, we know that the properties are clearly preserved; otherwise, we simply need to show that all packets in $BUFFER$ at positions before $C\_position$, $B_{C\_position}$ included, have laxities strictly inferior to L_{C_packet}. We know that $B_{C\_position}$ is not the last packet in $BUFFER$, otherwise we would be in the first subcase. And the call to the set of instructions INSERTION_1 is only possible if

$$L_{B_{C\_position+1}} > L_{B_{C\_position}} + 1 \tag{5.8}$$

Then, from Implication 1, we know that for any p packet in $BUFFER$ which precedes or is equal to $B_{C\_position}$, $r(p)$ precedes $B_{C\_position}$, because of the discontinuity in the L coordinate of the virtual laxity vectors after $C\_position$; and the result follows, since $L_{C\_packet} > L_{B_{C\_poistion}}$.

In the third case, the next stage is reached after a call to the set of instructions labeled EQUALITY, then properties hold, since the decreases of the coordinates of the virtual laxity vectors done in those instructions are consistent with the properties. In the fourth case, we assume that the next stage comes just after a call to the set of instructions

labeled INSERTION_2: all properties trivially hold since packets are simply inserted in $BUFFER$ according to their location order in the data structure; the location order is a total order in this case.

In the last case, the next stage is reached just after a call to the DROP instructions. Then, Property 3 is clearly preserved, and Property 2 is not affected for the packets that are located after g, the packet dropped. We need to show that Property 2 holds for C_packet and the packets located before g (those packets are inserted in $BUFFER$ during the dropping). Let p be one of those packets. If $r(p)$ was located after G before the dropping occurred, then Property 2 is preserved, since the decreases in the coordinates of the virtual laxity vectors are done consistently with Property 2. Now, if $r(p)$ precedes or is equal to g, then the packet located just before $r(p)$ becomes the new $r(p)$ after the dropping. This is true since the group virtual laxities are consecutive between p and $r(p)$, thus increasing them by one, automatically promotes the packet just before $r(p)$ as the new $r(p)$; and we know that p and $r(p)$ are different before the dropping, since p is located before g. Thus, the packet located just before $r(p)$ is well-defined. We have then showed that all properties hold at the next stage of the M-Drop procedure if they hold at the current stage, and since we showed that they are not affected by the scheduling procedure and the increase in time, they hold from the current stage on. They clearly hold at the very beginning of the scheduling process, and all results follow.

We thus completed the proof that $D^{min-max}$'s data structure has Poperties 1, 2 and 3. We now investigate the implications of those properties for the MOSTO optimality of the scheduling-dropping policy. We divide our investigation into two parts: we first show that the scheduling decisions are consistent with the MOSTO no-regret scheduling criterion specified in Theorem 2.3, and then proceed to show that the dropping procedure is such that a set strongly equivalent to $S_k(D^{min-max})$ is retained in each time slot k. We now consider the first part, and therefore consider that the M-drop procedure has been completed. From, what was done in the previous part, we know that the last packet of $BUFFER$ has the largest laxity among packets in $BUFFER$, and has its group virtual laxity equal to its laxity. Let m^* be the smallest position among the positions m such

that: B_m has a group virtual laxity equal to its laxity, and all packets preceding B_m in the buffer have laxities less than or equal to that of B_m, then we make the following claim:

Claim 5.1 $\{BUFFER[1], \ldots, BUFFER[m^*]\} = \Phi_s(BUFFER)$.

Proof. We first note that, from the definition of m^*, for any packet p preceding B_{m^*} in $BUFFER$, $r(p)$ precedes or is equal to B_{m^*}. Also the L group virtual laxities of the packets at the positions $1, \ldots, m^*$ of $BUFFER$ are consecutive, since otherwise, by Property 2, there would be a packet preceding B_{m^*} which would satisfy the definition of B_{m^*}, which is a contradiction.

For any packet k such that $1 \leq k \leq |BUFFER|$,

1. If $k > m^*$, then $L_{B_k} > L_{B_{m^*}}$, thus

$$l(B_k) \;>\; l(B_j) \quad \text{for any } j \leq m^* \tag{5.9}$$

Let s be the position of $r(B_k)$ so that $r(B_k) = B_s$. Property 3 implies that

$$L_{B_s} - s + 1 \;\geq\; L_{B_1} \tag{5.10}$$

Thus,

$$l(B_k) - s + 1 \;\geq\; L_{B_1} \tag{5.11}$$

Thus, since only packets in the first s positions can have laxity smaller than or equal to $L_{B_s} = l_{B_k}$.

$$l(B_k) - |\{p \in BUFFER : l(p) \leq l(B_k)\}| \;\geq\; L_{B_1} - 1 \tag{5.12}$$

2. Now let us assume that $k \leq m^*$, and keep the same definition for s as that of the previous case. Then, we already noted that $s \leq m^*$. Also, the consecutivity of the group virtual laxities for packets in positions $1, \ldots, m^*$ gives

$$L_{B_s} - s + 1 \;=\; L_{B_1} \tag{5.13}$$

Thus,

$$l(B_k) - s + 1 \;=\; L_{B_1} \tag{5.14}$$

We then show that if $s \neq m^*$ then

$$s \;>\; |\{p \in BUFFER : l(p) \leq l(B_k)\}| + 1 \tag{5.15}$$

Note that either $l(B_s) > l(B_k)$, or there exists $m \leq s$ such that $l(B_m) > l(B_s)$: this is true since otherwise s would be m^*, which is not the case. Thus, from Equation (5.14)

$$l(B_k) - |\{p \in BUFFER : l(p) \leq l(B_k)\}| \;\geq\; L_{B_1} \tag{5.16}$$

But, we know that

$$L_{B_{m^*}} - m^* \doteq l(B_{m^*}) - |\{p \in BUFFER : l(p) \leq l(B_k)\}| \;=\; L_{B_1} - 1 \tag{5.17}$$

Thus, $l(B_{m^*}) = l^*$, where l^* is defined as in Equation (2.8), and the result follows.

□

The following claim is a corollary to Claim 5.1:

Claim 5.2 *The first class 1 packet in the buffer, if any, is in* $\Phi_s(\overline{BUFFER})$.

Proof. The result can be proved by using an argument similar to the one developed for the proof of Claim 5.1 restricted to packets of class 1.

□

Claim 5.3 *If there is a class 1 packet in* $\{B_1, \ldots, B_{m^*}\}$, *then* B_1 *is a class 1 packet.*

Proof. The second part of Implication 1 basically says that if the first packet of the buffer is a class 2 then B_{m^*} is a class 2 packet, and so are all packets preceding it in the buffer. We simply need to remark that $r(B_1)$ is a class 2 packet, and so is any packet p preceding it in the buffer, if any, and any packet preceding $r(p)$, and thus by induction

all packets up to B_{m^*}.

□

We conclude from Claims 5.1, 5.2, and 5.3 that in each time slot k, $D^{min-max}$ schedules a packet in $\Phi_m(Q_k(D^{min-max}))$.

It now remains to show that the dropping procedure preserves a set strongly equivalent to $S_k(D^{min-max})$. Once again, we consider two cases: in the first case, we assume that a class 1 packet g is dropped. Then clearly, just before the dropping instructions, all packets located before g in the data structure, *C_packet* included, are class 1 packets, otherwise a packet of class two would have been dropped before reaching g: this can be seen using an argument like the one developed in the proof of claim 5.3. Also, we know that $K_g = L_g = l(g)$. Let p be a packet located before g in the data structure just before the dropping instruction, and let $\tilde{r}(p)$ be the class 1 packet in $BUFFER$ such that $K_{\tilde{r}(p)} = l(p)$. We know such a packet exists from Implications 1 and 2. If $\tilde{r}(p)$ precedes or is equal to g, then $l(p) \leq l(g)$. On the other hand, if $\tilde{r}(p)$ is after g, the class virtual laxities of class 1 packets are consecutive between p and $\tilde{r}(p)$,

$$
\begin{aligned}
l(p) &= K_g + \text{(the number of class 1 packets between } g \text{ and } \tilde{r}(p)) + 1 && (5.18)\\
&\leq K_g + \text{(the number of packets between } g \text{ and } \tilde{r}(p)) + 1 && (5.19)\\
&\leq L_g + \text{(the number of packets between } g \text{ and } \tilde{r}(p)) + 1 && (5.20)\\
&\leq L_g + \text{(the number of packets between } g \text{ and } r(p)) + 1 && (5.21)\\
&\leq l(p) && (5.22)
\end{aligned}
$$

We used the fact that $\tilde{r}(p)$ necessarily precedes or is equal to $r(p)$ to make a substitution between those packets in the inequalities above. Thus, all packets between p and $\tilde{r}(p)$ are class 1 packets. Also, we know that there cannot be a discontinuity of the group virtual laxity of packets located between p and $\tilde{r}(p)$ (both included), since we showed that, at the positions of discontinuity in that coordinate of the virtual laxity vector, the packet on the side closer to the head of $BUFFER$ has laxity larger than or equal to the laxity

of the packets that precede it, if any, and has both coordinates of its virtual laxity vector equal to its laxity. Clearly, such a packet cannot be between p and $\tilde{r}(p)$. Therefore, we use the consecutivity of both coordinates of the virtual laxity vectors between g and $\tilde{r}(p)$, and we have $L_{\tilde{r}(p)} = K_{\tilde{r}(p)}$. Then we see, from Property 2, that all packets between $\tilde{r}(p)$, and $r(\tilde{r}(p))$ are class 1 packets, and $L_{r(\tilde{r}(p))} = K_{r(\tilde{r}(p))'}$. We can inductively repeat the argument until we reach a class 1 packet v such that $L_v = K_v = l(v)$, and all packets located before v have laxity less than or equal to that of v and are class 1 packets. Then, this shows that the class 1 packets already buffered plus *C_packet* form an unschedulable set, since the cardinality of the set of packets with laxity smaller than or equal to $l(v)$ is $l(v)+1$, that is, $l^* = -1$. Thus, one packet of the set mentioned above has to be dropped. The packet g dropped is such that the remaining set of class 1 packets is equivalent to the one that would have been obtained by dropping a class 1 packet with smallest laxity: this is true since the packets from the set above which are retained still covers the first $l(v)$ positions of $BUFFER$, thus the next $l(v)$ time slots, the current one included. And the proof is completed for the first case.

We now consider the second case: a class 2 packet g is dropped. Then, we know that the set of packets in the data structure is not schedulable. By the definition of g, we know that just before the dropping, g is the first packet with laxity equal to its group virtual laxity, and, by Property 2, we know that there exists a class 2 packet g' with $L_{g'} = l(g')$ which is either equal to g or located after g such that all packets located between the first class 2 packet of the data structure (first in the location order) and g' (both border packets included) are class 2 packets and have laxity smaller than or equal to $L_{g'}$. We assume that there are l of those packets. Then, dropping g is equivalent to dropping the smallest laxity class 2 packet among those packets, since the $l-1$ packets remaining from the set described above, if any, can cover the relative time slots $l(g') - l + 2, \ldots, l(g')$. And Properties 1 and 3 tell us that the set of packets retained in the buffer after the dropping is schedulable. Thus, we showed that in both dropping cases above, the packet dropped is such that a strongly equivalent set is retained. Lemma 7.1 in [1] shows that a

strongly equivalent set can be constructed sequentially, considering one packet at a time. And Claim 2.2 follows.

5.2 Proof of Claim 2.3

The technique for the proof is similar to the one used to prove Theorem 2.5 from Theorem 2.3. We use an inductive approach. The induction hypothesis is:

$$\mathcal{P}(N) \ : \ \begin{cases} \text{The } D^{min-max} \text{ policy is SPTO} \\ \text{for an } N \text{ class model} \end{cases}$$

Then, a technique similar to the one already used in the proof of Theorem 2.5 is applied. We assume now that we have an $N+1$ class model. First we consider that the two lowest priority classes are grouped in a super-low-priority class, and apply the $D^{min-max}$ algorithm as defined for a N class case. Since it is assumed in the M-Insert subprocedure that two lowest priority packets with equal first coordinated of their virtual laxity vector are ordered arbitrarily, it does not affect the SPTO property of the $D^{min-max}$ policy for the N class model, if the lowest priority class is split into two subclasses and the scheduling policy is extended to $N+1$ classes. Thus, the $D^{min-max}$ scheduling policy as defined for a $N+1$ class model is SPTO when restricted to an N class model obtained from the previous one by grouping the two lowest priority packets in a super low priority class. And of course, we use the fact that MOSTO scheduling policies are still MOSTO even when vacation time slots exist in the system. We use a reasoning similar to the one presented in the proof of Theorem 2.5. Properties 1, 2, and 3 presented in that proof are still valid. Implications 1 and 2 also hold. The following claim is simply an extension of Claim 5.1 to the general multiclass case. The proof is similar. It is not given in this thesis.

Claim 5.4 $B_1 \in \Phi_{SPTO}(BUFFER)$.

The following fact can also be shown using an argument similar to the one used to prove its validity in the two class case.

Fact: For any $i \in \{1, \ldots, N+1\}$, if $D^{min-max}$ drops a packet class-i packet p, then no $(N-1)$-SPTO (for the N class model) continuation of $D^{min-max}$ can schedule all packets in $BUFFER \cup p$ with priority higher than or equal to that of p; and dropping p is equivalent to dropping a smallest laxity packet among those packets.

Now, if for the sake of an argument by induction, we assume that for some k, the scheduling and dropping decisions of $D^{min-max}$ prior to k do not preclude $D^{min-max}$ from being N-SPTO, one can see that the induction hypothesis $\mathcal{P}(N)$, Claim 5.4 and the fact above imply that the dropping and scheduling, if any, done by $D^{min-max}$ in time slot k would not preclude $D^{min-max}$ from being N-SPTO. Of course, for $k = 0$, the induction hypothesis is true. The result follows.

5.3 Proof of Claim 2.4

We do not provide a detailed proof of the claim here. It can be done using a reasoning similar to the one presented in the Section 5.2. The only modification consists in grouping the two highest priority classes in a super-low-priority class, in order to reduce the $N+1$ class case to a particular instance of an N class case, and then apply the recursive definition of an NTO scheduling policy and the implications of Properties 1 and 2 presented above: they imply that $q_1 \in \Phi_{NTO}(S_k(D^{nested}))$ for each time slot k in which the buffer is not empty. The no regret nature of the dropping is then guaranteed by an argument similar to the one given above for $D^{min-max}$.

CHAPTER 6

SYNTHESIS

The results presented in this thesis show that it is possible to optimize prioritizing in a setting where throughput optimality is required. We showed that it is possible to do so causally while significantly reducing the buffer occupancy. Furthermore, we provided several algorithms that perform different levels of prioritizing while being essentially no more costly, in term of their complexity, than a simple implementation of the EDF scheduling policy. We stress that the optimal dropping performed by those algorithms makes them more efficient than the nondropping EDF scheduling policy. Also, the characterization of all the scheduling policies that satisfy the criteria presented in this thesis makes it possible to develop more efficient algorithms.

The result in Lemma 3.3 is particularly interesting since it implies one particular application of our characterizations. If the model was extended to include a particular class of extremely high-priority packets (control packets for example) that must be served as soon as they are available, Lemma 3.3 indicates that it is still possible to achieve the optimality criteria on the remaining classes of packets subject to the scheduling of urgent packets. Also, the hierarchical nature of the criteria we considered here makes it possible to apply the criteria only to a particular subtraffic. We summarize our results: we characterized all causal scheduling and scheduling-dropping policies that maximize the throughput in a sample-pathwise manner. Among those policies, we characterized the ones that minimize the buffer occupancy in a sample-pathwise manner. We then proceeded to study how much prioritizing could be achieved subject to throughput optimality. We defined two novel and important criteria: nested and static priority throughput optimality. The first criterion consists of recursively optimizing the throughput of a smaller set of higher

priority subject to the optimalization of the throughput of the next largest set of higher-priority packets. The second criterion consists of recursively optimizing the throughput of a class of packets subject to the optimization of the throughput of the next higher priority class. We presented simple algorithms that perform those optimizations while minimizing the buffer occupancy over all causal throughput optimal scheduling policies.

REFERENCES

[1] B. Hajek and P.-F. Seri, "On causal scheduling of multiclass traffic with deadlines," UIUC preprint 1997.

[2] S. Baruha, G. Koren, B. Mishra, A. Raghunathan, L. Roisier, and D. Sasha, "On-line scheduling in the presence of overload," *IEEE Symposium on Foundations of Computer Science*, pp. 100-110, 1991.

[3] J. Liebeherr, D. E. Wrege, and D. Ferrari, "Exact admission control for networks with a bounded delay service," *IEEE/ACM Transactions on Networking*, vol. 4, no. 6, pp. 885-901, December 1996

[4] R. L. Cruz, "A calculus for network delay, Part I: Network elements in isolation," *IEEE Trans. Inform. Theory*, vol. 37, no. 1, pp. 132-141, January 1991.

[5] V. C. S. Lee, J. K. Y. Ng, K.-Y. Lam and S.-L. Hung, "Performance studies of transmitted real-time MPEG-I video in ATM networks," in *Proceedings LCN'96* , pp. 153-160, October 1996.

[6] J. M. Peha and F. A. Tobagi, "Evaluating scheduling algorithms for traffic with heterogeneous performance objectives," in *Proceedings IEEE GLOBCOM*, pp. 21-27, December 1990.

[7] T.-L. Ling and N. Shroff, "Scheduling real-time traffic in ATM networks," in *Proceedings IEEE INFOCOM '96*, pp. 198-205, March 1996.

[8] T.-L. Ling and N. Shroff, "Scheduling real-time traffic in ATM networks," submitted to *IEEE/ACM Transaction on Networking*, 1996.

[9] R. Chiopalkatti, J. F. Kurose, and D. Towsley, "Scheduling policies for real-time and non-real-time traffic in a statistical multiplexer," in *Proceedings IEEE INFOCOM '89*, pp. 774-783, Ottawa, April 1989.

[10] E. L. Lawler, *Combinatorial Optimization: Networks and Matroids.* New York: Holt, Rinehart and Winston, 1976.

[11] C. H. Papadimitrou and K. Steiglitz, *Combinatorial Optimization: Algorithms and Complexity.* Englewood Cliffs, NJ: Prentice Hall, 1982.

To appear. CISS 99

CAPACITY AND RELIABILITY FUNCTION PER UNIT COST FOR WSSUS FADING CHANNELS

Vijay G. Subramanian and Bruce Hajek *

ABSTRACT

This paper summaries our recent work on the application of the concepts of capacity per unit cost and reliability function per unit cost to models of fading channels. The cost is taken to be the energy, or a certain fourth moment functional that is sensitive to the burstiness of the signals, and is related to the ambiguity function of the input signal. The channel is assumed to be a Wide-Sense Stationary and Uncorrelated Scattering (WSSUS) channel.

1. INTRODUCTION

A prominent feature of wireless media is time-varying multipath fading. The fading channel is very different from the additive Gaussian noise (AGN) channel. If the channel changes rapidly, then we are forced to adopt non-coherent techniques[1] for reliable communication. Another important fact is that for pure fading channels, the output signal has mean zero for any input signal. Thus, the input signal only affects the second order statistics and higher order statistics of the output. In contrast, the input signal directly affects the mean of the output signal for AGN channels. Owing to these differences, principles of signal design used for additive Gaussian noise channels do not directly apply to fading channels[2]. Thus, it is necessary to take a fresh look at the problem instead of directly applying the principles used for AGN channels.These ideas constitute the motivation behind this paper. The objective of this paper is to contribute to the better understanding of the capacity of multipath fading channels.

*The authors are with the Coordinated Science Laboratory and the ECE Department, University of Illinois, Urbana, IL 61801. Email:{ vgsubram,b-hajek}@uiuc.edu. This work was supported by a Motorola Fellowship, by the US Army Research Office under Grant DAAH05-95-1-0246, and by the National Science Foundation under contract NSF NCR 93-14253.

[1]Refer to Kennedy[12], Marzetta and Hochwald[14] and Telatar and Tse[17] for examples of non-coherent receiver structures that achieve capacity.

[2]Refer to Bigleri, et al.[4] for a discussion on this.

2. WIRELESS CHANNEL MODELS

Wireless channels can be thought of as time-varying linear channels. The channel model can be represented by

$$y(t) = \int h(t,\tau)u(t-\tau)d\tau + n(t), \qquad (1)$$

where $u(t)$ is the input, $h(t,\tau)$ is the time-varying channel transfer function, $n(t)$ is white Gaussian noise, and $y(t)$ is the output of the channel. It is assumed that $h(t,\tau)$ for fixed τ is a wide-sense stationary (WSS) process, i.e., $E[h(t,\tau)] = \mu(\tau)$, and $E[h(s,\tau)h^*(t,\tau)] = R_H(s-t,\tau)$. It is also assumed that $h(t,\tau)$ is a Gaussian random process. We also have $h(t,\tau)$ uncorrelated for different values of τ. This is called uncorrelated scattering(US). Most stochastic models of wireless channels combine these two features (see Fleury, et al.[7]) and consider what are called WSSUS fading channels. We denote the multipath delay-spread by $T_{\max}$, the doppler-spread by F_d, and the coherence time by $T_{\text{coherence}}$. We refer to Clarke [6], Bello [2], and Proakis [16, Chapter 14] for a detailed treatment of WSSUS fading channels.

3. CAPACITY OF FADING CHANNELS

Even though wireless channels have been used for a long time, they are not as well understood as the additive Gaussian noise channel. We refer the reader to Bigleri, et al. [3] for a detailed survey of capacity related results on fading channels.

Abou-Faycal, et al.[1] compute the capacity of the average energy constrained discrete-time memoryless Rayleigh fading (DTMRF) channel. Marzetta and Hochwald[14] generalize this channel to include multiple transmit antennas, multiple receive antennas, and channel memory. They model the channel fading coefficient as constant over intervals of T symbols in duration, and independently chosen for different intervals. They compute the capacity numerically and show that, for large values of T, the capacity is close to the capacity with the receiver knowing the channel perfectly.

Broadband channels are a special case of channels with a large number of degrees of freedom. Gallager[10] in his seminal work discusses energy limited channels, i.e., channels where the energy per degree of freedom is very small. Restricting the input to binary signals he computes the reliability function [9]. Considering the case of infinitely many degrees of freedom he shows that the reliability function can be computed exactly for all rates if there is a finite capacity per unit energy. He also characterizes the conditions under which capacity per unit energy is infinite. Gallager also considers channels with non-binary inputs but with a single zero-energy signal and proves that using binary signaling is optimal in the limiting case of infinitely many degrees of freedom. Telatar [18] specializes Gallager's results to the Rayleigh fading channel. He shows that with very high bandwidths and at high SNRs, the Rayleigh fading channel has the same capacity per unit energy as an AWGN channel with the same SNR and bandwidth. Verdú[19] concentrates on capacity per unit energy cost instead of the reliability function per unit energy and generalizes Gallager's idea of capacity per unit energy to include more general cost functions. He derives a simple characterization for the capacity per unit cost for memoryless channels. He shows that the capacity per unit cost, C_{cost}, is determined by a simple functional maximization. Under Gallager's assumption that there is a single input with zero cost, Verdu shows that

$$C_{\text{cost}} = \sup_{x:\text{cost}(x)\neq 0} \frac{D(P_{Y|X=x}||P_{Y|X=0})}{\text{cost}(x)}, \quad (2)$$

where $D(\cdot||\cdot)$ is the Kullback-Liebler distance between measures and the coding scheme to be followed is to use on-off keying with the on input being the maximizer of the above functional, say x_{max}, and the off input being the zero cost input. To approach C_{cost}, the on input is to be used with vanishingly small probability. As an example he shows that the capacity per unit energy cost of the discrete-time Gaussian fading channel is the same as the AWGN channel.

Kennedy [12] considers the capacity per unit time of diffuse WSSUS fading channels. He constrains the input signals to be of the form $u(t)e^{-j2\pi\Delta_i t}$, $1 \leq i \leq M$, i.e., the signals form an M-ary FSK signaling set with the signal $u(t)$ as a design parameter. Gallager and Médard[8] analyze a system with diffuse WSSUS fading. They partition the broadband channel into frequency bins and constrain the input in each bin at each time to be as follows: $E[|X|^2] \leq \epsilon$, and $E[|X|^4] \leq \alpha\epsilon^2$. This models DS-CDMA type signals where the power is allocated uniformly over time and bandwidth. They show that the mutual information per unit time between the input and the output is now upper-bounded by a constant times ϵ. The interpretation of this bound is as follows: as the spread factor increases without bound, ϵ which is inversely proportional to the spread factor, decreases to zero, and therefore, the mutual information per unit time between the input and the output decreases to zero. Telatar and Tse[17] consider specular multipath channels with time-varying delays of the various multipath components, in the case of no intersymbol interference.

Note that there can also be the case where there is no knowledge of the statistics of the channel. Lapidoth, et al. [13] give an extensive treatment of such channels.

4. CAPACITY CALCULATIONS

We consider using the WSSUS fading channel in blocks of time T' units long where $T' \geq T + T_{\text{coherence}} + T_{\text{max}}$ with T being the duration of the input signal. We also assume that $u(t) = 0$ if $t > T$. Assuming that we code across blocks of time T' units long, these assumptions imply that we

use the channel in a memoryless manner. We can write the output of the channel as given in Equation (1) where $h(t,\tau)$ is a complex-valued, zero-mean Gaussian process with $E[h(t,\tau)h^*(s,v)] = R_H(t-s,\tau)\delta(\tau - v)$, and $n(t)$ is white, complex-valued Gaussian noise with one-sided power spectral density σ^2. We also constrain the input waveforms to have finite energy.

Equation (1) can be written in the following manner

$$y(t) = s_{out}(t) + n(t), \quad 0 \le t \le T', \tag{3}$$

where, given u, $s_{out}(t)$ is a complex-valued, zero mean Gaussian random process independent of $n(t)$ with covariance function given by $\Sigma(s,t) = E[s_{out}(s)s^*_{out}(t)]$. Let $\{\lambda_i\}_{i=0}^{\infty}$ be the eigenvalues of the covariance operator, Σ. Then by Mercer's theorem [15, p. 379] and Proposition VI.D.4 [15, p. 421] we have that

$$\begin{aligned} D(u) &\triangleq D(P_{Y|U=u}||P_{Y|U=0}) \\ &= \sum_{i=0}^{\infty} \phi(\lambda_i), \end{aligned} \tag{4}$$

where

$$\phi(\lambda) = \frac{\lambda}{\sigma^2} - \log(1 + \frac{\lambda}{\sigma^2}). \tag{5}$$

4.1. Capacity per unit energy cost

In this section we derive the capacity per unit energy cost of WSSUS fading channel. Since there is only one input function, namely, $u(t) \equiv 0$, having zero energy, we can apply Verdú's [19] results directly. Thus, the capacity per unit energy cost denoted by C_E, is given by

$$C_E = \sup_{u \neq 0} \frac{D(u)}{E(u)},$$

where u denotes the input waveform and $E(u) = \int_s |u(s)|^2 ds$ is the energy of the waveform u. We have that

$$D(u) = \frac{\mathrm{Tr}(\Sigma)}{\sigma^2} - \sum_{i=0}^{\infty} \log(1 + \frac{\lambda_i}{\sigma^2}).$$

Therefore,

$$\begin{aligned} \frac{D(u)}{E(u)} &= \frac{G_H}{\sigma^2} - \frac{\sum_{i=0}^{\infty} \log(1 + \frac{\lambda_i}{\sigma^2})}{\sum_{i=0}^{\infty} \frac{\lambda_i}{\sigma^2}} \frac{G_H}{\sigma^2} \\ &\le \frac{G_H}{\sigma^2}, \end{aligned}$$

and fixing an arbitrary but nonzero signal u we have that $\lim_{a\to\infty} \frac{D(u\sqrt{a})}{E(u\sqrt{a})} = \frac{G_H}{\sigma^2}$. Therefore, we have established that

$$C_E = \frac{G_H}{\sigma^2}, \tag{6}$$

and the capacity can be approached by using any nonzero input as the on input for the on-off keying scheme with the energy tending to infinity and the average energy constraint tending zero. We should note that C_E is exactly the same as the capacity per unit energy of an AWGN channel with the same gain and noise characteristics. This is exactly as expected from the results of Jacobs [11], Kennedy [12], and Telatar and Tse [17].

4.2. Capacity per unit fourth-moment cost

Define $J_C(u) = \sum_{i=0}^{\infty} \lambda_i^2$. We can use $J_C(u)$ as a cost function and we can apply Verdú's [19] result to obtain the capacity per unit cost. We can show the following simple expression for $J_C(u)$,

$$J_C(u) = \int_{\nu} \int_{\tau} |\chi(\nu,\tau)|^2 \psi_H(\nu,\tau) d\tau d\nu, \tag{7}$$

where $\chi(\tau,\nu)$ is the symmetric ambiguity function [5] of the signal $u(t)$ which is defined as follows

$$\chi(\nu,\tau) = \int_{-\infty}^{\infty} u(t+\tau/2)u^*(t-\tau/2)e^{-j2\pi\nu t} dt, \tag{8}$$

and where $\psi_H(\tau,\nu)$ is given by

$$\psi_H(\nu,\tau) = \int_f \int_t S_H(f,t) S_H(f+\nu, t+\tau) dt df.$$

Note that $J_C(u)$ is a fourth-order cost function. Note also that $J_C(u)$ captures both time and frequency aspects of the signal. In fact, it can be shown that $J_C(u) \le G_H^2 \int |u(t)|^4 dt$. We can also show that $J_C(u) \le G_H^2 \int |U(f)|^4 df$ where $U(f)$ is

the Fourier transform of $u(t)$. We can, thereafter, show the following

$$\sup_{u \neq 0} \frac{D(u)}{J_C(u)} = \frac{1}{2\sigma^4}. \tag{9}$$

We can state the following theorem by applying Verdú's [19] results.

Theorem 4..1 *The capacity per unit fourth-moment cost, C_J, of the WSSUS fading channel is given by*

$$C_J = \frac{1}{2\sigma^4}. \tag{10}$$

As a consequence we have for any input random process U,

$$I(U;Y) \leq \frac{1}{2\sigma^4} E[J_C(U)], \tag{11}$$

where Y is the output random process and the expectation is carried out with respect to the measure of U.

A very important point that we must emphasize here is that Kennedy [12] defines the number of effective diversity paths, D to be the reciprocal of $J_C(u)$. In [12] u is the M-ary FSK waveform while here it is the on signal for on-off keying. Thus, D increasing without bound implies that $J_C(u)$ decreases to zero and the result of the error exponent going to zero in [12] is mirrored by the mutual information between the input and the output going to zero.

Before going to the next topic we state an important property of ambiguity functions. The volume invariance property [5] states that

$$\int\int |\chi(\nu,\tau)|^2 d\tau d\nu = \chi(0,0)^2 = (\int |u(t)|^2 dt)^2 = E(u)^2. \tag{12}$$

Note also that $|\chi(\nu,\tau)| \leq \chi(0,0) = E(u)$.

4.3. Other related results

Using the bound given in Theorem 4..1 in the full version of this paper we derive a bound on the capacity of Direct-Sequence Code Division Multiple Access (DS-CDMA) like signals and show that under certain conditions the capacity tends to zero as the spreading increases for diffuse WSSUS channels. Considering specular WSSUS channels we show that the capacity of DS-CDMA like signals for high spreading gains is inversely proportional to the number of paths when the paths have almost equal energy. We also derive capacity per fourth moment cost results in the multiuser case. Assuming joint coding of the users we show that the sum capacity is upper bounded by a function of the cross-ambiguity functions of the different inputs. Specializing to the case of independent users we show that the sum capacity behaves similar to the single user capacity. We qualitatively show that multiuser interference yields a sum capacity much lower than the sum of the single user capacities.

5. RELIABILITY FUNCTION CALCULATIONS

From Gallager[10] we can write the following expression for the reliability function per unit cost,

$$\hat{E}_r(\hat{R}) = \max_{0 \leq \rho \leq 1} \left(\sup_{u:cost(u) \neq 0} \frac{(-(1+\rho)\log E^{H_0}[\Lambda^{1/(1+\rho)}])}{cost(u)} \right) - \rho \hat{R}, \tag{13}$$

where $cost(u)$ is the cost associated with input u, $E^{H_0}[\cdot]$ is an expectation taken with respect to the measure generated by the **0** input, and Λ is the likelihood ratio of the input u with respect to the **0** input.

5.1. Reliability per unit fourth moment cost

We now specialize the results of the previous section using the fourth moment cost function we used before, i.e., $cost(u) = J_C(u)$. We can show that $\tilde{E}_r(\tilde{R})$ is given as follows

$$\tilde{E}_r(\tilde{R}) = \frac{1}{2\sigma^4} \beta(2\sigma^4 \tilde{R}), \tag{14}$$

where

$$\beta(r) = \begin{cases} \frac{1}{2} - r & 0 \leq r \leq \frac{1}{4} \\ (1 - \sqrt{r})^2 & \frac{1}{4} \leq r \leq 1 \end{cases}. \tag{15}$$

It is interesting to note that we get an expression for the reliability function which does not depend on the channel except through σ^2. We also have another instance where the exact reliability function

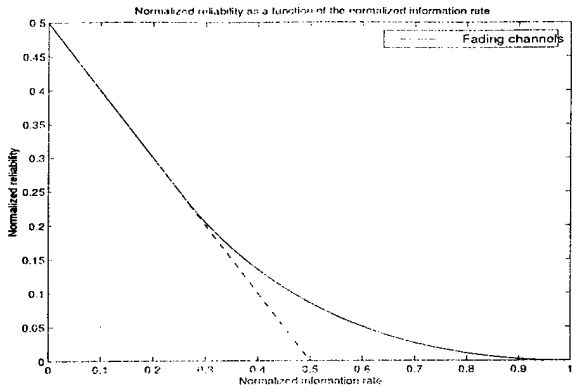

Figure 1. Normalized reliability per unit fourth moment cost as a function of normalized information rate.

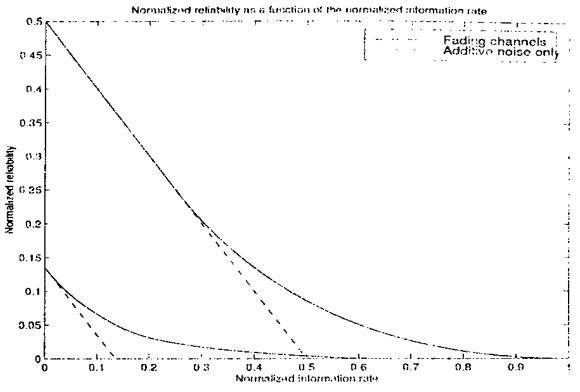

Figure 2. Normalized reliability per unit energy as a function of normalized information rate.

can be computed. It is also interesting to note, as indicated in Figure 1, that the shape of the reliability function is exactly that of the infinite bandwidth Additive White Gaussian Noise channel [9, p. 381].

5.2. Reliability per unit energy

We now specialize the results of the previous section using the energy of the input, i.e., $E(u)$, as the cost function. For this case we derive a channel independent upper bound. We show that

$$\hat{E}_r(\hat{R}) \leq \frac{G_H}{\sigma^2}\gamma(\frac{\hat{R}\sigma^2}{G_H}), \tag{16}$$

where

$$\gamma(r) = \max_{0\leq\rho\leq 1}\sup_{\alpha>0} \frac{\log(1+\alpha)+(1+\rho)\log(1-\frac{\alpha}{1+\alpha}\frac{1}{1+\rho})}{\alpha} - \rho r. \tag{17}$$

The bound shows that Kennedy's optimized system reliability function [12, p. 125] is indeed an upper bound for the infinite bandwidth WSSUS fading channel reliability function. Numerically evaluating the upper bound we compare it with the reliability per unit energy for the AWGN channel in Figure 2. As expected from our discussion above, Figure 2 is similar to Figure 5.5 in Kennedy [12, p. 125]. We can now reiterate all the conclusions about fading channels in Kennedy's book [12, pp. 125-135].

REFERENCES

[1] I. C. Abou-faycal, M. Trott, and S. Shamai. The capacity of discrete-time memoryless rayleigh fading channels. *submitted to IEEE Trans. on Information Theory*, 1997.

[2] P. A. Bello. Characterization of randomly time-varying linear channels. *IEEE Trans. Commun. Syst.*, CS-11:360–393, Dec 1963.

[3] C. Bigleri, J. Proakis, and S. Shamai (Shitz). Fading channels: Information-theoretic and communication aspects. *IEEE Trans. on Information Theory*, 44(6):2619–2692, Oct 1998.

[4] E. Bigleri, G. Caire, and G. Taricco. Coding and modulation for the fading channel. In *Proc. IEEE Vehicular Technology Conf.*, May 1997.

[5] R. E. Blahut. *Theory of Remote Surveillance.* Manuscript to be published by Cambridge University Press.

[6] G. C. Clarke. A statistical theory of mobile radio reception. *Bell System Tech. Journal*, 47:957–1000, 1968.

[7] B. H. Fleury and P. E. Leuthold. Radiowave propagation in mobile communications: An overview of european research. *IEEE Communications Magazine*, 34(2):70–81, Feb 1996.

[8] R. Gallager and M. Médard. Bandwidth scaling for fading channels. In *Proc. International Symposium on Information Theory'97 (ISIT)*, page 471, Ulm, Germany, June-July 1997.

[9] R. G. Gallager. *Information Theory and Reliable Communication.* John Wiley and Sons, New York, 1968.

[10] R. G. Gallager. Energy limited channels: Coding, multiaccess, and spread spectrum. *Tech.*

Report LIDS-P-1714, LIDS, MIT, Cambridge, Mass., November 1987.

[11] I. Jacobs. The asymptotic behavior of incoherent M-ary communication systems. *Proc. IEEE*, pages 251–252, Jan 1963.

[12] R. S. Kennedy. *Fading Dispersive Communication Channels.* Wiley Interscience, New York, 1969.

[13] A. Lapidoth, P. Narayan, and M. S. Pinsker. Reliable communication under channel uncertainty. *preprint*, Dec 1997.

[14] T. Marzetta and B. Hochwald. Capacity of a mobile multiple-antenna communication link in a rayleigh flat-fading environment. *Submitted to IEEE Transactions on Information Theory*, 1997.

[15] H. V. Poor. *An Introduction to Signal Detection and Estimation.* Springer-Verlag, 1988.

[16] J. G. Proakis. *Digital Communications.* McGraw-Hill, New York, 3rd. edition, 1995.

[17] E. Telatar and D. Tse. Capacity and mutual information of broadband multi-path fading channels. *preprint*, 1997.

[18] I. E. Telatar. Coding and multiaccess for the energy limited rayleigh fading channel. Master's thesis, Dept. of Electrical Engineering and Computer Science., Massachusetts Institute of Technology, Cambridge, Mass., 1986.

[19] S. Verdú. On channel capacity per unit cost. *IEEE Trans. Info. Th.*, IT-34(5):1019–1030, Sept. 1990.